The Thesis Committee for Carolyn Leigh Powell
Certifies that this is the approved version of the following Thesis:

Deriving Rock Strength from MSE and Drilling Data

APPROVED BY
SUPERVISING COMMITTEE:

Eric van Oort, Supervisor

Pradeepkumar Ashok, Member

Deriving Rock Strength from MSE and Drilling Data

by

Carolyn Leigh Powell

Thesis

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

May 2019

Dedication

I would like to dedicate this work to my parents. To my father who inspired me to enter the oil and gas field, and who has supported and encouraged me in all of my academic and professional pursuits. To my mother who instilled in me a sense that any goal is achievable and nothing is impossible. To both of them for their unending love and support.

Acknowledgements

I would like to thank Dr. Eric van Oort and Pradeepkumar Ashok for their guidance and supervision on this work. Their support was instrumental to my success and I am grateful for their mentorship throughout this academic endeavor. I would also like to thank Matthew Prinz for his contributions to the development of the random forest algorithm presented in this work, and the UT RAPID research team. Finally, I would like to thank my family and friends, in particular my fiancée Simon and my parents whose love and support make my success possible.

Abstract

Deriving Rock Strength from MSE and Drilling Data

Carolyn Leigh Powell, M.S.E.

The University of Texas at Austin, 2019

Supervisor: Eric van Oort

As the industry works to reduce costs and enhance completion techniques, engineered completions have emerged as a promising method to improve hydraulic fracturing efficiency. However, the method remains cost and labor intensive, limiting widespread adoption. A cost effective and easily implemented approach to engineered completions is needed. A data driven method utilizing Mechanical Specific Energy (MSE) has been proposed to denote relatively homogeneous sections of rock along the wellbore using only commonly available drilling data. This work investigates the MSE based engineered completions methods presented in the literature, and argues that the parameters which drive the MSE term may be more compelling indicators of rock heterogeneity. Additionally, automated pattern recognition methods to identify characteristic parameter response behaviors to either a rock strength or drilling efficiency change are explored. A Random Forest algorithm for defining characteristic parameter behaviors is presented and discussed, indicating promise for Machine Learning methods to define a library of parameter responses to energy changes that can be automatically detected while drilling the well, with positive implications for both completions design and drilling optimization.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

Technological advancements in the oil and gas industry have made significant improvements on the cost and efficiency of the average drilling operation. Today, approximately one third of the cost of on an onshore unconventional well may be due to drilling costs, while the remaining two thirds are devoted to the completion. As completion designs become more complex, typically with higher density placement of stages and perforations, the industry is looking for ways to enhance the well economics while reducing production risk due to inefficient completions. This thesis explores the potential use of commonly acquired drilling data to denote "like rock" sections of the wellbore for engineered completion design, as well as for drilling optimization

A typical completion follows a geometric design that divides the lateral of a horizontal well into evenly spaced stages. Within each stage are placed evenly spaced perforations from where the hydraulic fracture treatment will propagate fractures into the formation. No consideration is given to heterogeneity of the rock along the wellbore when designing the geometric placement of stages and perforations (Skinner, Van Domelen, & Grieser, 2016). A company may then apply the same geometric design to every well in an area under development in what is known as the "factory method." Experience has shown that this approach does not render consistent fracture propagation results across all stages and thus results in inconsistent production performance. However, the factory method lends itself to efficient equipment procurement and operational execution. In a given stage, the rock strength may vary and result in one or two fractures preferentially propagating into the weakest sections of rock. The remaining perforations see less of the hydraulic fracturing fluid and thus do not receive an effective treatment. If the rock is fairly

homogeneous within a stage, the perforations may propagate more evenly, resulting in a better stimulation overall within that stage. Without designing for rock heterogeneity, completing a suite of wells with a geometric completion design means companies may be missing opportunities for completion optimization and improved production (Logan, 2015).

Engineered completions utilize well log and geological data to design a completion specific to the rock encountered in each wellbore. A variety of companies currently offer this engineering service; however, the process tends to be time consuming and expensive. To see an economic benefit, a company must apply the engineered completion approach to every well in their field. Therefore, the significant cost must be justified by an estimation of overall production enhancement across the field, which carries uncertainty and risk. From the perspective of operations, implementing an engineered completion program means the equipment and materials requirements for each well will vary, making procurement planning and execution more complex, and introduces additional operational risk due to the unique procedural requirements for each well. The adoption of engineered completions in their current form has been slow despite the potential economic gains due to these hurdles. Therefore, a cost justified, easily implemented, and consistently effective approach to engineered completions is needed to allow these efficiency gains to be unlocked.

A drilling data driven solution to this problem has been proposed based on Mechanical Specific Energy (MSE) (Logan, 2015; Ouenes, Dirksen, Paryani, Rehman, & Bari, 2017). MSE has been used successfully in drilling operations to increase rate of penetration (ROP) and optimize drilling performance (Dupriest & Koederitz, 2005). It captures the total amount of energy consumed by the drilling system per the volume of

rock removed (Teale, 1965). Changes in the energy consumed can be due to two primary changes in the system: an efficiency change or a rock strength change. In the case of an efficiency change, the energy consumed in the system that is not contributing directly to breaking new rock has changed. In the case of a rock strength change, the energy required to break new rock has changed. The drilling parameters used to calculate MSE are collected in real time on every well drilled, and its calculation is fairly straight forward, as described in Chapters 2 and 3. If MSE can be utilized as an indicator for rock strength changes while drilling, the process for identifying homogeneous or "like rock" sections would be accelerated and only require common drilling data. The interpretation of "like rock" for stage placement could happen in relatively real time while the well is being drilled.

The work presented in this thesis explores the concepts presented in the literature using MSE as an indicator of "like rock". The drilling parameters that define the MSE term are analyzed in the context of efficiency, as the separation of efficiency and rock strength changes is necessary when interpreting the MSE term for "like rock" sections. Therefore, the traditional applications of MSE for drilling optimization presented in the literature are examined as the foundational work for parameter relationships under different efficiency states. The drill off curve is a good visual representation of different zones of operational efficiency, and can help guide drilling operation with the intention of keeping the bit within the highest efficiency state possible (Dupriest & Koederitz, 2005). This not only maximizes ROP, but allows the efficiency state to remain relatively constant, potentially making it easier to observe rock strength change responses in the MSE. When efficiency and rock strength changes are taking place simultaneously, interpretation of the MSE term for drilling or completion purposes becomes more difficult.

The data analytics methods described in this thesis were used to analyze the MSE term and its associated drilling parameters with the intention of identifying trends or behaviors that may distinguish a rock strength change. The MSE term as well as the its driving parameters (ROP, WOB, RPM, and torque) were analyzed together, revealing different parameter behaviors at moments of increasing MSE. The different parameter behaviors may be responses to specific efficiency or rock strength influences. Therefore, focus was placed on parameter response behavior analysis and pattern recognition methods applied to the drilling parameters instead of the MSE term directly. Automated pattern recognition algorithms were developed and applied to the drilling data examined in this in study, showing some promise in the ability to recognize unique influences in the responses. The results presented in this thesis indicate that an optimized machine learning model applied to a large data set may reveal categorical parameter responses specific to a rock strength change or efficiency change that can be recognized in a real time drilling operation as the parameter data streams are collected at surface. However, the direct interpretation of MSE to rock strength along the wellbore proved more complex than the literature appears to imply.

Chapter 2: Literature Review

MSE has been used in the industry to successfully improve drilling operational performance and is now being leveraged to inform engineered completion design decisions. For drilling optimization, parameter adjustments can be made by monitoring MSE to enhance drilling efficiency in real time, and then reviewed during post drill analysis for re-design on future wells. The concept has also shown promising results for completions which are increasingly moving toward engineered completion design. The mechanical energy required to fail a volume of rock is influenced by various geomechanical properties and in situ rock states, therefore some correlation is assumed to exist between the MSE behavior and rock heterogeneity. A survey of the current literature on both drilling and completions applications for MSE is presented, as well as a discussion on the current limits of this work.

2.1 Mechanical Specific Energy

The concept of MSE for drilling rock was first introduced by Teale in 1965. He defined MSE as the ratio of the work done versus the volume of rock removed. The initial derivation was based on tricone roller bits which dominated the industry at the time Teale published his work. However, he argued that the same fundamental components of work are performed when drilling rock using any bit type. The theoretical minimum energy required to remove a volume of rock is not dependent on the bit or system being used, but is related to the mechanical properties of the rock being drilled (Teale, 1965).

The drilling process can be broken down into two components: thrust and rotation. The thrust component consists of the weight of the drill string being applied at the cutting structure and thus indenting the cutters into the rock (Teale, 1965). Teale equated this to

an indenter, or laboratory tool used to measure rock hardness. When an axial force is applied to the cutting structure, it results in crushing of the rock face at the point of indentation creating small cracks locally within the solid surface. He argues that the size of the wellbore is relatively small compared to the formation rock and therefore the formation can be treated as a semi-infinite solid, as in an indentation test. Rotation of the bit applies a torsional force that sweeps across the bottom hole, "cutting" pieces from the rock surface. The two mechanisms are effectively taking place simultaneously but for conceptual simplification can be pictured as alternating, with the indentation first followed by the rotation. The combination of the two results in the removal of various sized rock fragments that are broken free from the brittle rock surface, as opposed a finite volume removed directly via displacement of the cutter (Teale, 1965). Therefore, Teale argues that the combined process is more akin to "breakage" than "cutting". Relating the energy applied to the size of the fragment removed from a semi-infinite brittle solid by crushing or breakage is thus the foundation of the MSE relationship and was a novel contribution by Teale at the time. The terms cutting, chipping, breaking, and drilling are used interchangeably in the following sections and are intended to describe the general mechanism of removing rock through drilling operations.

Using drilling operational parameters as inputs, Teale defined MSE as shown in Equation 1.

Equation 1

$$MSE = \frac{WOB}{A_{bit}} + \frac{120\pi * RPM * Torque}{A_{bit} * ROP}$$

MSE is given in terms of pressure (psi in field units) (Pessier & Fear, 1992). The thrust term is determined by the weight on bit (WOB) applied versus the projected circular

surface area of the bit (A_{bit}). The equation models the drilling process as the combined axial and torsional forces applied evenly across the entire circular bottom hole rock face, and thus can be equated to pressure (Teale, 1965). The rotary term is determined by the ratio of the rotational velocity (or rotations per minute, RPM) and the torque applied to the cutting structure versus the area of the bit and rate of penetration (ROP). The ratio of ROP to RPM is defined as the penetration per rotation (or depth of cut, DOC), defined by Equation 2, and can be substituted back into Equation 1 as shown in Equation 3 (Teale, 1965).

Equation 2

$$DOC = \frac{ROP}{RPM}$$

Equation 3

$$MSE = \frac{WOB}{A_{bit}} + \frac{120\pi * Torque}{A_{bit} * DOC}$$

The MSE relationship thus equates the amount of torque applied to remove a rock layer that has thickness equal to the calculated DOC (Teale, 1965). The rotary term is again equated to a rotational force moving evenly over the circular bottom hole surface area in one full rotation.

Teale's foundational work inspired further applications and studies of specific energy for drilling applications. Further research built upon his original concepts and introduced some notable modifications and enhancements for laboratory and field work. Pessier and Fear (1992) recognized that torque as measured at surface did not accurately represent the torque at the bit due to frictional losses in the drill string. To account for this

in a way that can be easily implemented, they introduced a bit specific coefficient of sliding friction, μ, to estimate torque in the absence of reliable downhole torque measurements. They modeled the bit as the flat end of a circular shaft, based the idea that the forces applied at the bit are evenly applied across the circular surface area of the bottom hole. They derived the relationship shown in Equation 4, which can be substituted for torque into Equation 1. In field applications with PDC bits, the coefficient of sliding friction is typically assumed constant at 0.5 (Pessier & Fear, 1992).

Equation 4

$$Torque = \mu \frac{WOB * D_{bit}}{36}$$

If a mud motor is used, torque may be estimated using the mud motor design parameters. A mud motor facilitates bit rotation through a motor stator arrangement, driven by the drilling mud pressure. Replacing the measured torque at surface with the ratio of the maximum torque (T_{max}) vs. maximum differential pressure (ΔP_{max}) ratings of the mud motor, and multiplying this ratio by the differential pressure across the motor (ΔP) gives a reasonable estimate for torque at bit in the presence of a mud motor (Logan, 2015). The mud motor introduces a simultaneous downhole rotation at the bit, in addition to the drill string rotation. This bit rotation can be maintained under sliding conditions as well, or when the drill string is not rotating but still drilling ahead. RPM is thus equal to the combined surface RPM (N) and the mud motor RPM, which is the product of the mud flow rate (Q) and the revolutions per barrel rating of the motor used (K_n). Mud motor modifications to Equation 1 result in Equation 5 (Logan, 2015).

Equation 5

$$MSE = \frac{WOB}{A_{bit}} + \frac{120\pi * (N + Q * K_n) * \left(\frac{T_{max}}{\Delta P_{max}} * \Delta P\right)}{A_{bit} * ROP}$$

Armenta (2008) introduced a hydraulic term to the MSE equation to account for the effects of bit cleaning. If the bit is efficiently cleaned while drilling, less energy is being dissipated to regrind or break up cuttings already removed from the rock surface. Thus, the cutters are more efficiently engaging with new rock (Armenta, 2008). Armenta called the modified MSE term Drilling Specific Energy (DSE), shown in Equation 6.

Equation 6

$$DSE = \frac{WOB}{A_{bit}} + \frac{120\pi * RPM * Torque}{A_{bit} * ROP} + \frac{1{,}980{,}000 * \lambda * HPP_{bit}}{ROP * A_{bit}}$$

The third term in the DSE equation is Armenta's contribution, where the ratio of hydraulic horsepower at the bit (HPP_{bit}) to bit area is equal to the horsepower per square inch at the bit, or HSI. The 1,980,000 multiplier is a unit conversion factor and λ is a dimensionless constant based on the bit diameter (Armenta, 2008).

2.2 Drilling Optimization Applications

The energy required to remove a volume of rock is not the only energy consumed within the drilling system. Therefore, an efficiency can be defined to measure the overall drilling performance given the geology and drilling parameters. Energy application through rotation and thrust is measurable at the surface, however transfer of that energy must travel through the drill string to the bit before being applied to the rock face. The rock

being drilled will determine how much work must be done at the bit, which in turn is reflected in the energy input requirements at surface to maintain a desired ROP. This reciprocal relationship can be leveraged to define a drilling efficiency state, which can then be monitored to ensure efficiency is maximized at all times. This concept has been utilized in the field with repeated success for the optimization of drilling operations, however identifying and quantifying specific inefficiencies in the system has proven difficult.

2.2.1 Drilling Efficiency

The theoretical minimum energy required to remove a volume of rock, or MSE, should be approximately equal to the rock's resistance to ductile failure. Once the energy applied to the rock face reaches this value, the rock will fail and produce a chip or cutting. Therefore, it is implied that MSE is correlated to the strength of the rock being drilled, represented by compressive strength, as well as the various rock properties that influence compressive strength (Teale, 1965).

Teale performed laboratory drilling experiments using a variety of bit types and measured the parameters used to calculate MSE. The rock strength properties of the specimens drilled were known, and the experiments were conducted in a controlled laboratory setting at atmospheric pressure. Thrust, or WOB, was increased in a controlled manner and the impact on the MSE term was recorded. With increasing WOB, MSE trended towards a minimum that was approximately equal to the unconfined compressive strength (UCS) of the rock being drilled (Teale, 1965). Teale concluded that when the bit is operating at peak efficiency, the MSE term should be approximately equal to the UCS of the rock.

Dupriest and Koederitz (2005) expanded upon this concept to define Mechanical Efficiency while drilling, or "Drilling Efficiency." They recommended that this term be used to qualitatively assess drilling operations in real time. Drilling Efficiency (DE) is defined in Equation 7, where MSE_{min} is the theoretical minimum MSE achievable, or UCS. For practical application, MSE_{min} may be the minimum observed MSE through optimization of drilling input parameters (Dupriest & Koederitz, 2005).

Equation 7

$$DE = \frac{MSE_{min}}{MSE} * 100$$

The DE term implies that there is a combination of drilling input parameters governing the amount of energy applied to the drilling system where at peak efficiency the energy applied is primarily being consumed by the bit-rock interaction for cutting new rock (Dupriest & Koederitz, 2005). The input parameters are those contained within the MSE equation, including ROP, RPM, Torque, WOB, and bit diameter. Some knowledge of the geomechanical properties of the rock being drilled is required to estimate the rock strength while drilling and thus compare it to the MSE value to calculate a drilling efficiency. Dupriest and Koederitz argue that the overall trend of the MSE while drilling is more important than the actual calculated value and therefore direct comparison is not necessary. Therefore, MSE_{min} defined as the minimum achievable MSE over a given drilling section, as observed during WOB and RPM adjustments, can be used as the baseline MSE from which to observe DE changes. Understanding the baseline MSE in a given formation thus eliminates the need for knowledge of the actual rock strength, as deviations from the baseline are indicators that a new inefficiency has been introduced. This qualitative

indication of the efficiency state can alert drillers that optimization of the drilling operation is possible (Dupriest & Koederitz, 2005).

When a new lithology or section is to be drilled, a common field test may be performed called a “drill off test” to determine the optimal combination of WOB and RPM to achieve maximum ROP over the section. A drill off test is performed by increasing the WOB in stages and measuring the corresponding ROP response. The two are then plotted together in what is known as a “drill off curve.” The drill off curve typically takes the form of a slanted S shape curve, as shown in Figure 2.1. The curve shows an initial non-linear response prior to sufficient engagement of the cutting structure with the rock, a linear response where efficient drilling is taking place, and a final non-linear response, the beginning of which is referred to as the “founder” point (Dupriest & Koederitz, 2005).

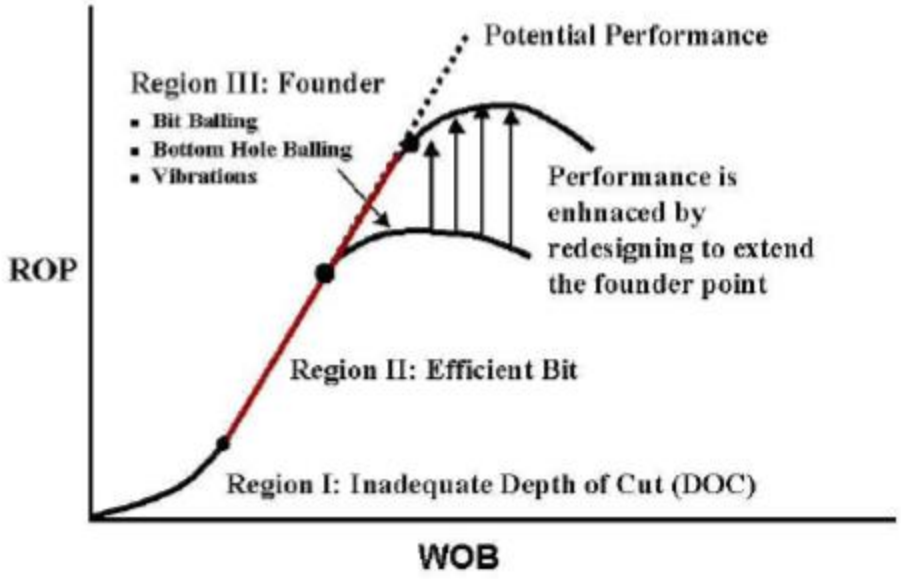


Figure 2.1: Idealized Drill Off Curve (Dupriest & Koederitz, 2005)

With the results of the drill off test, the driller would set WOB to be just below the founder point to achieve maximum ROP in this section, and take the resulting MSE as the baseline MSE. Additionally, the test may be repeated with different RPM settings to determine the optimal combination of RPM and WOB to maximize ROP. The behavior of MSE while drilling can be related to the three distinct regions of the drill off curve. The founder point is considered the maximum achievable ROP with the current system parameters, including RPM, mud properties, and rock properties. After the founder point is reached, the drilling efficiency suffers and the ROP ultimately starts to fall which in turn increases MSE (Dupriest & Koederitz, 2005).

When evaluating the drill off curve over a period of drilling time, one must assume that the properties of the rock being drilled, the hydraulic properties of the system, the frictional losses in the system, and the bit wear state remain constant. With these assumptions, the linear region, or Region II per Figure 2.1, of the drill off curve will exhibit a constant and minimum MSE value as this is the most efficient drilling region (Dupriest & Koederitz, 2005). In an efficient drilling state, any increase in axial energy results in a proportional increase in torsional energy and ROP to maintain a constant value of MSE. Similarly, an increase in RPM does not increase torque but results in a proportional increase in ROP, keeping MSE constant. Following the drill off curve in this region, an increase in WOB will result in a linear increase in ROP due to a proportional increase in DOC. The linear behavior between WOB and ROP implies that when drilling efficiently, the only requirement to increase ROP is to increase the energy input through WOB, while RPM is held constant. If the mud flow rate were increased changing the bit hydraulics, there would be no effect on ROP as the bit is already efficient (Dupriest & Koederitz, 2005). The characteristic linear slope of ROP vs. WOB corresponds to the bit specific coefficient of

sliding friction as described by Pessier and Fear (1992) and previously defined. A change in bit design will change the characteristic linear behavior of WOB and ROP when the new bit design is in an efficient drilling state (Dupriest & Koederitz, 2005). Theoretically, the minimum MSE should remain the same, however the new bit design may be less efficient overall, therefore an increase in the baseline MSE would be observed.

With the given MSE baseline, deviations from this MSE value will indicate the introduction of inefficiency in the system. DE is defined previously as the proportion of the energy input into the system that is being used to break the rock. While in Region II, any additional energy added to the system will be consumed by the bit-rock interaction. If MSE increases with added energy, all or some of the additional energy is not being used to break new rock, and is being consumed elsewhere in the system through a source of inefficiency. By monitoring MSE in real time, the driller can identify changes in the efficiency state that may indicate an issue and bring the system back into its baseline efficient state, or back into Region II of the drill off curve (Dupriest & Koederitz, 2005).

At low values of WOB, the cutters are not sufficiently penetrating the rock to result in efficient drilling. This state corresponds with Region I of the drill off curve, per Figure 2.1. When the DOC is very low, energy is consumed at the bit primarily in the form of friction between the cutters and the rock (Zhou, et al., 2012). In some cases, a low DOC may result in a drilling dysfunctional state called “whirl”. Evidence shoes that whirl can be overcome by increasing WOB and reducing RPM (Dupriest & Koederitz, 2005). Whirl, as well as other drilling dysfunction, will be described in more detail in Section 2.2.3.4.

Beyond the founder point, efficiency limiters begin to take over. Additional WOB no longer results in a proportional ROP response as some of the applied energy is now being consumed by other mechanisms, or inefficiencies, that were not present in Region

II. These inefficiencies may include hydraulic cleaning issues or the various forms of drilling dysfunction. The efficiency of the system is compromised beyond the founder point, although a brief continued rise in ROP may be possible. However, this rise will no longer follow the characteristic linear trend observed in Region II. Monitoring MSE along with the ROP vs. WOB trend together provides a more detailed assessment of the efficiency state of the system, and what may be influencing the occurrence of a founder point (Dupriest & Koederitz, 2005). We will explore some of the factors that influence when a founder point occurs and how MSE monitoring can result in an expanded window for efficient drilling in the following sections.

2.2.2 Case Studies: MSE to Improve Drilling Performance

Using a real time calculation of MSE, drillers can implement the drilling optimization technique presented by Dupriest and Koederitz described in the previous section. Changing parameters and observing the MSE response allows the driller to determine a baseline MSE to be used as a relative performance indicator. As MSE increases from the baseline, drilling efficiency is potentially reducing and therefore additional ROP may be achievable by adjusting drilling parameters. If the founder point causing the inefficiency is limiting the parameter adjustments, the system may need to be redesigned to move the founder point and increase the maximum achievable ROP (Dupriest & Koederitz, 2005).

In 2004, ExxonMobil implemented what they call the Fast Drill Process (FDP) which uses real-time MSE monitoring to improve ROP (Dupriest, Witt, & Remmert, 2005). Dupriest, Witt, and Remmert presented examples of the effectiveness of the FDP workflow in the Qatar North Field where different dysfunctional states were recognized and

remediated through MSE monitoring. In one case, minor bit balling was taking place at high WOB, and would worsen with WOB increase. With the basic assumption that maximum WOB delivers maximum ROP, drillers maintained the highest WOB where ROP was still at an acceptable level while experiencing some bit balling. While monitoring MSE, the drillers observed that MSE was relatively high. By lowering WOB, MSE was reduced and ROP increased, and thus efficiency increased. Lowering WOB to increase ROP is counter intuitive without the assistance of MSE monitoring. WOB will typically induce an increase in ROP in the absence of bit balling or other founder, or when the system is efficient. The field engineers were later able to evaluate the MSE response on this well after drilling was completed and justify a bit redesign to improve hydraulics at the bit. This allowed for higher WOB to be applied on the next wells without causing bit balling issues and thus a further increase in maximum ROP. The bit redesign changed the founder point for the system, allowing higher WOB and ROP with a linear response (Dupriest, Witt, & Remmert, 2005).

Another example where MSE monitoring was used to improve ROP performance was again in the Qatar North Field. As WOB was increased, an increase in efficiency (or decrease in MSE) was observed, indicating whirl at the lower values of WOB. As previously mentioned, whirl is a vibrational state that results from insufficient DOC. The increase in WOB caused a steep increase in ROP and decrease in MSE as the whirl was remediated. The ROP increase was due to the DE change, not strictly due to the additional energy input through WOB, which was relatively small. As WOB continued to increase, MSE continued to drop, indicating a continued improvement from the whirl state. The existence of whirl was confirmed through logging while drilling (LWD) vibration data. However, as MSE continued to drop with increasing WOB, indications of whirl in the

LWD data had already ceased. This may be an example where MSE was more sensitive to a vibration dysfunction state than the LWD vibration data. The LWD data was able to accurately diagnose the nature of the vibration as whirl, but MSE monitoring allowed the driller to continue optimizing the system (Dupriest, Witt, & Remmert, 2005). This highlights the potential for combining additional downhole data sources with MSE monitoring to enhance the drillers ability to respond to inefficiencies.

2.2.2.1 Combining MSE with Other Methods to Improve Drilling Performance

There are many examples within the literature where MSE monitoring has led to ROP improvements. The examples presented in this section are three cases where MSE monitoring was expanded upon with either additional data, through data analytics across a suite of well data, or through automated driller response to improve the overall drilling performance. The examples presented are diverse in their applications, but are based on the founding principles of MSE and MSE monitoring from the work of Dupriest and Koederitz, as previously discussed.

Pinto and Lima (2016) improved drilling operations in deepwater Brazil, using downhole data to calculate and monitor MSE. A downhole calibration tool behind the bit also took vibration measurements and relayed them to surface in real time. The use of downhole data to calculate MSE provided a better estimate of the bit state in downhole conditions, and therefore a more accurate MSE value representative of the bit-rock interaction. Combining this with downhole vibration data enabled quick and accurate diagnosis of the dysfunctional state when drilling became inefficient, making the drillers better informed when determining a mitigation plan (Pinto & Lima, 2016).

Alsubaih, et al. (2018) analyzed the drilling parameter data and system states of wells in southern Iraq. Parameters from initial wells such as hydraulics, rock strength, bit condition, and parameter inputs like WOB and RPM were compared with MSE. Linear regressions were run on each parameter with respect to MSE to estimate parameter specific coefficients that were then used to predict ROP. Using the resulting parameter model, they then optimized the parameters to predict a maximum achievable ROP in offset wells. When the offsets were drilled using the pre-optimized parameters, actual vs. predicted MSE was plotted to determine if the chosen parameters were in fact maintaining a high level of DE (Alsubaih, Albadran, & Alkanaani, 2018). However, the coefficients are likely equipment, driller, and geology specific making this method difficult for broad implementation.

Zhao, et al. (2017) built an automated driller response module based on Dupriest and Koederitz work for real time MSE monitoring. The system monitors the drilling parameters and MSE, looking for the onset of founder points. When a founder point is believed to be reached, the system initiates a logic sequence, implementing staged adjustments based on the initial hypothesis of the cause of the founder. For example, if the founder is believed to be stick slip dysfunction, the system will reduce WOB incrementally until the parameters stabilize and MSE is minimized. They combine this with downhole vibration data to aid in the identification of the founder source (Zhao Fei, Wang Haige, Cui Meng, Zhang Huabei, & Pang Huiwen, 2017).

2.2.3 Discussion on Limitations of DE Implementation

Teale showed that the minimum MSE achievable corresponded to a maximum efficiency state where the MSE value was approximately equal to the UCS of the rock being drilled. Therefore, the compressive strength can be used as a relative comparison for

the efficiency state of the system while drilling. As MSE is a cumulative measure of the energy consumed by the system as a whole at any point while drilling, it would be incorrect to take MSE as a direct reading of the compressive strength of the rock at any given drilling time (Dupriest & Koederitz, 2005). A variety of energy sinks exist throughout the system to inflate the MSE term above the minimum energy required to fail the rock. Acknowledging and understanding these energy sinks is important when drawing conclusions on the state of the system from the MSE and DE values as they may be interdependent and/or taking place concurrently and thus difficult to isolate from each other. Responding effectively to reduce the impact of one of the various inefficiencies or energy sinks in the system requires accurately diagnosing the source and magnitude of the inefficiency. Here we will describe some of practical considerations to be aware of when comparing MSE to UCS and monitoring DE, as well as the physical states that impact the MSE value and thus DE, illustrating the difficulty in isolating specific physical phenomenon within the MSE term.

2.2.3.1 Practical Considerations when Comparing MSE to UCS

Teale's experiments suggest that MSE is a measure of compressive strength. However, Teale also argues against this conclusion, highlighting that MSE should be used with caution when drawing conclusions on rock strength. The crushing mechanism while drilling results in irregularly sized fragments that do not equate exactly to the volume swept by the cutting structure with each rotation (Teale, 1965). Therefore, the volume of rock removed is not precisely linear to the DOC or WOB, as modeled by the MSE relationship and the drill off curve. However, the linear approximation is sufficient for drilling optimization applications, evidenced by the data trending towards linearity under efficient

cutting conditions (Dupriest & Koederitz, 2005). Rock failure under drilling conditions may also be dependent on how the cutter indentation is applied to the rock, including cutter orientation or design and the flatness or smoothness of the rock surface (Teale, 1965). In general, Teale observes that his recorded MSE values are highly dependent on the experimental set up. The energy required to execute the drilling process, or MSE, is thus not an absolute measure of rock strength or compressive strength. Rather, MSE and compressive strength are both functions of rock strength, therefore one can assume that MSE and compressive strength are correlated in some way (Teale, 1965).

Notably, Teale conducted his experiments at atmospheric pressure. Therefore, a confining pressure, or pressure differential between the wellbore and the rock pore space, was not applied. The rock strength may be changed under confining conditions. While drilling, the confining conditions are dictated by the difference between the in-situ rock pore pressure and the equivalent circulating density (ECD) of the drilling mud. Thus, changes in ECD can influence the confining pressure seen at the bit. The measure of rock strength under these conditions is typically more than the UCS value as drilling is most often performed in an overbalanced state. The rock strength under these conditions is referred to as the confined compressive strength (CCS). CCS can be estimated from UCS using Equation 8 (Majidi, Albertin, & Last, 2016).

Equation 8

$$CCS = UCS + \Delta P \left(\frac{1 + \sin\theta}{1 - \sin\theta}\right)$$

CCS is depended on the confining pressure, ΔP, and the internal friction angle, θ, of the formation being drilled. In order to estimate CCS while drilling, UCS and internal friction

angle data must be available, either from laboratory core measurement or sonic log interpretation. UCS and internal friction angle are found from sonic logs using empirically derived relationships, examples of which are shown in Equation 9 and 10. This is a commonly used relationship for Gulf of Mexico Miocene and Pliocene shale lithologies, where V_p is sonic compressional wave velocity (Majidi, Albertin, & Last, 2016).

Equation 9

$$UCS = 0.43 * V_p^{3.2}$$

Equation 10

$$\theta = 1.532 * V_p^{0.5148}$$

Because the as drilled MSE value is typically higher than the estimated UCS of the rock, Dupriest and Koederitz recommend a constant Eff_m term applied to the calculated MSE value while drilling of 35%. This is based on a typical range of baseline efficiencies they observed in their work which were between 30 and 40% (Dupriest & Koederitz, 2005). In reality, the efficiency state of the drilling system is constantly changing due to a variety of factors that may be acting on the system simultaneously. However, a constant multiple of 35% to scale the MSE value down to a range where trends in UCS can be compared to trends in MSE is useful, as it allows the rig personnel to more easily interpret the MSE and drilling efficiency trends (Dupriest & Koederitz, 2005).

2.2.3.2 Surface vs. Downhole Drilling Parameters

In addition to the inherent inaccuracies when comparing MSE and rock strength, as described by Teale, the parameters from which MSE is calculated also contain inaccuracies

in their measurement. The MSE term is most often calculated using surface measured parameters, which include ROP, WOB, RPM and torque. RPM is a relatively reliable surface input and ROP is measured based on traveling block movement, therefore the error in these as measured at surface is relatively low. Due to tortuosity and frictional forces along the wellbore, some of the weight of the string is lost in these contact forces and does not reach the bit. Torque at the bit is similarly affected by torsional friction along the wellbore that reduces the applied torque at bottom. The surface measured reaction torque is not only the bit-rock interaction torque, but also the torque the entire drill string must overcome due to friction to rotate the string. Because MSE is highly sensitive to torque fluctuations, inaccuracies in torque introduce noise in the MSE term that can mask correlations with other well influences, like rock strength (Islam, et al., 2018). To obtain a more accurate representation of torque and WOB, a torque and drag model can be applied to the wellbore trajectory to estimate friction losses and adjust the torque and WOB measured at surface (Kerkar, Hareland, Fonseca, & Hackbarth, 2014). Various torque and drag models exist, and selection should be based on the ability to apply the model in real time to produce real time adjusted parameters. This approach will provide a better estimate of the downhole condition at the bit, but will again be an estimate. The best method for measuring torque and drag is thus at the bit via downhole sensors. However, the cost and logistics of measuring downhole parameters and transmitting that data to surface in real time for MSE monitoring are not necessarily justified on every well.

2.2.3.3 Bit Wear

As the bit drills ahead, the bit will typically experience wear to some degree. The cutting structure may wear in a gradual fashion or dull over time, introducing more friction

and less sharp cutting action. Bit dulling changes the way the DOC is applied and thus the nature of the bit-rock engagement. One or more bit teeth (or cutters) may chip or break, or the bit body may experience wear or damage. The impact on ROP and other drilling parameters will depend on the nature of the bit wear. When the wear state changes, the MSE value will change as the drilling parameters react to the change (Waughman, Kenner, & Moore, 2002).

It is very difficult to predict bit wear or estimate the current bit wear state while drilling. Bit wear is also highly variable and therefore the impact on MSE is also variable and difficult to diagnose. How the bit is damaged, when it is damaged, and how severely it is damaged or worn depends on the rock being drilled and how the bit is being used to drill it. Therefore, bit wear influences on MSE are very difficult to distinguish from other sources of inefficiency, and often develop simultaneously with other inefficient states like drilling dysfunction or vibration. Dulling may induce a gradual MSE change, while chipping may cause a step change in MSE, both of which can be misidentified as another form of inefficiency. As Teale observed, compressive strength and MSE are not synonymous but correlated by rock strength and dependent on the nature of the cutting. Bit wear induces a change in the nature of cutting and thus produces a change in MSE and DE (Waughman, Kenner, & Moore, 2002).

2.2.3.4 Bit Hydraulics and Hole Cleaning

For drilling to continue, the bit requires continuous exposure to a fresh rock face where the process can repeat itself. The energy required to break a solid into smaller pieces was found to increase as the size of the solid is reduced (Teale, 1965). This concept is important to the discussion of DE due to the presence of previously produced cuttings or

chips at the bottom of the hole. These cuttings are removed by hydraulic energy applied through circulation of drilling mud, sweeping the newly produced cuttings out from in front of the cutting structure and keeping the bit "clean". If cuttings are not removed efficiently, their presence in front of the cutting structure will result in the dissipation of energy applied at the bit to break these finite volume solids into smaller pieces, reducing the overall DE (Armenta, 2008).

When WOB is increased, the increase in DOC results in a larger volume of cuttings being produced. If the bit hydraulics are insufficient to remove the cuttings volume, a dysfunctional state can develop known as bit or BHA balling. Balling is when cuttings accumulate around the bit and/or BHA, creating a buffer between the cutting structure and the rock face. In the most severe cases, the cuttings may conglomerate at the bit in a way that creates a barrier between the cutting structure and the rock, such that little to no further cutting of new rock is taking place. This is most often seen in softer lithologies that contain large amounts of clay or shales (Dupriest, Witt, & Remmert, 2005). Intuitively, a severe reduction in efficiency would occur as ROP would be hindered by the lack of engagement with new rock. Energy is thus used in rotating the bit and the cuttings caked on and around the bit at the bottom of the hole, with little bit-rock engagement. From a system design perspective, a change in bit hydraulics, through either maximum achievable flowrates or bit design, may widen the window for efficient drilling, allowing for higher ROP in areas prone to this dysfunction (Dupriest, Witt, & Remmert, 2005). In some areas, the clays are susceptible to expansion, sometimes referred to as "gumbo", which may make the section incompatible with water based muds. However, in the context of MSE monitoring and parameter reactions in real time, mud flow rates and ROP adjustments are the only options a driller has immediately available.

A founder point may be reached when hydraulics are no longer efficient, moving the drill off curve into Region III. The corresponding MSE value will increase under these conditions as drilling efficiency has decreased. If the driller recognizes the founder point as bit balling, reducing WOB will reduce ROP and reduce the volume of cuttings produced. This may remediate the hole cleaning issue and bring the system back into an efficient cutting state. Alternatively, the mud flow rate may be increased to improve bit hydraulics and cuttings removal (Dupriest, Witt, & Remmert, 2005).

2.2.3.5 Drilling Dysfunction

A variety of drilling dysfunctional states may produce a founder point. If the bit enters a dysfunctional state, accurate diagnosis of the dysfunction may be difficult but is critical for determining the correct course of action. Dysfunction may occur as one or a combination of the following forms: whirl, stick slip, bit bounce, or bit balling. Bit balling was discussed in the context of bit hydraulics in the previous section.

Whirl occurs when there is insufficient engagement between the cutters and the rock, resulting in an eccentric rotation of the BHA and drill string within the wellbore. Whirl can occur during both slide drilling and rotary drilling, and typically results in a spiraled wellbore path and/or an oversized borehole, both of which can lead to additional inefficiencies in the system. Severe whirl can cause high lateral loading as the BHA and bit slam against the walls of the wellbore while rotating off axis. Bit damage is commonly observed under whirl conditions, which is more immediately evident in the ROP response than gradual bit wear effects. In addition to potential bit damage effects on MSE, whirl places the drill string in a vibrational state that consumes energy at the expense of ROP, and results in an increase in the MSE value. An increase in WOB is the recommended

response to bring the string out of whirl (Dupriest & Koederitz, 2005). Whirl is observed most often within Region I of the drill off curve, where WOB is still low and bit engagement insufficient for efficient drilling.

Stick slip is a torsional vibration the bit will experience when there is not enough energy applied to fail the volume of rock the bit has engaged. Because the rock has not failed, the bit remains stationary while the string continues to rotate, building torque within the drill string. When the torque is sufficient to fail the rock, the string will release this torque suddenly. When the bit engages with the rock again, the energy input into the system has not changed, thus the process will repeat in the form of torsional vibration assuming the rock strength has not changed. This typically means the DOC is too high for the given applied energy and rock strength. Reducing WOB will reduce the DOC and volume the bit is attempting to fail with each "bite.". The energy required to fail the rock is thus reduced making the energy input into the system sufficient for instantaneous failure. Stick slip most often corresponds with Region III of the drill off curve, creating a founder point that limits the maximum WOB that can be applied (Dupriest & Koederitz, 2005).

While all of the described inefficiencies induce an increase in MSE, the magnitude of that increase is difficult to predict. Similar changes in DE may be due to two unrelated inefficiencies. Additionally, a change in DE may be due to a combination of inefficiencies evolving simultaneously. Gradual changes in maximum achievable DE are expected as the well gets deeper and more directional, due primarily to frictional losses. Therefore, DE monitoring alone does not diagnose the issue. The driller must interpret all of the information available at the time when they observe MSE diverge from the baseline, and decide on a course of action. Therefore, identifying the individual contributors to

inefficiency and quantifying their impact on MSE are major challenges, that if overcome can provide the driller with more accurate and effective corrective actions.

2.3 COMPLETIONS OPTIMIZATION APPLICATIONS

As previously discussed, a change in rock strength will change the bit-rock interaction state, and thus manifest a change in the MSE trend. If all other influences on MSE are held constant, the only change in MSE would then be due to a change in rock strength that the bit is encountering. However, at no time are all other MSE influences constant in a realistic drilling scenario. Therefore, to draw conclusions on downhole rock strength while drilling, the rock strength change must be isolated from all other MSE influences and inefficiencies in the system. If rock strength changes can be distinguished on a scale that is of interest to completions designs, wellbore heterogeneity can be determined from drilling data alone and applied to an engineered completion workflow. Companies are currently marketing such workflows for engineered completions services, some of whom have published on their proprietary methods. The methodologies presented in the literature to date are discussed here, as well as the potential error and complexity in these methods.

2.3.1 MSE for Engineered Completions

A variety of companies have seized upon the concept of deriving UCS from MSE to promote the use of MSE as a tool for engineered completions services. Engineered completions are uniquely designed completions specific to an individual well. A typical completion design consists of treatment stages of a standardized length and spacing along the entire lateral wellbore, or wellbore within the target formation. Each stage contains the same number of perforations, placed in a repeated and evenly spaced pattern around the

wellbore. The fracturing fluid is pumped into the perforations one stage at a time to produce and propagate fractures into the formation. A large area under development may receive the same completion design on every well. This is called the "factory method" for completions operations. While this allows for efficient treatment preparation and execution, the average effectiveness of completions across the entire field may not be optimized. Understanding rock strength differences along the wellbore allows for the completion design to specifically target "like rock", or areas with minimal heterogeneity in rock strength (Logan, 2015). It is understood that if one perforation is placed within a weaker rock, that perforation will consume more treatment fluid as it propagates faster into the formation at the given treatment pressure. This leaves less fluid to propagate the remaining perforations, resulting in a suboptimal treatment in these fractures. An engineered completion design allows for uniquely designed stage spacing and sizes, as well as perforation cluster spacings and patterns, with the intention of fracturing more like rock within each stage and enabling a more even hydraulic fracturing treatment across all perforations (Logan, 2015).

For the operator to see economic benefit from engineered completions, the design methodology must be applied to the entire field of wells. Each well in the field will thus be subjected to a unique completion design. Therefore, the process for engineered completions must be fast, reliable, and easily implemented to reduce economic risk and justify the expense (Logan, 2015). Some of the companies currently marketing engineered completions services that utilize MSE as a key input are listed below.

- C&J Energy Services
- FracGeo
- Drill2Frac

- Quantico QDrill

All employ the MSE term, calculated using drilling data, as an indicator of rock strength heterogeneity along the lateral or wellbore to inform a workflow that generates engineered completion recommendations in relatively real time. Each company has a proprietary work flow for data collection, cleaning, and modifying for calculation of MSE. Some use the MSE information along with other log data and geomechanical models to enhance their design decision process. Overall, the currently marketed services are based on MSE being an indicator of the strength of the rock being drilled.

2.3.2 Engineered Completion Methodologies using MSE

Logan (2015), representing C&J Energy Services, described using MSE as an analog for UCS and rock heterogeneity along the wellbore. Citing the need for a low cost and reliable engineered completion technique, Logan argues that his approach accomplishes this using common drilling data, simple data manipulation, and the MSE term. The MSE trend with depth is then used to design targeted stage placement for engineered completions. MSE is used as a qualitative indicator for UCS, under the assumption that the drilling state is constantly efficient (Logan, 2015).

MSE in Logan's methodology is calculated using the modified version of Teale's specific energy equation that accounts for a mud motor, Equation 5. For simplification, the drilling parameters are obtained through typical surface measurement. Some data cleaning and manipulation takes place to eliminate anomalies and spikes from the surface measurement (Logan, 2015). However, no external models are applied to the data to better estimate downhole conditions and no other data source is used to compliment the MSE interpretation. The workflow proposed by Logan takes the calculated MSE and splits the

values into finite ranges in magnitude. The MSE plot is then colored to show the MSE magnitude ranges along the wellbore. An example of the MSE log output is shown in Figure 2.2.

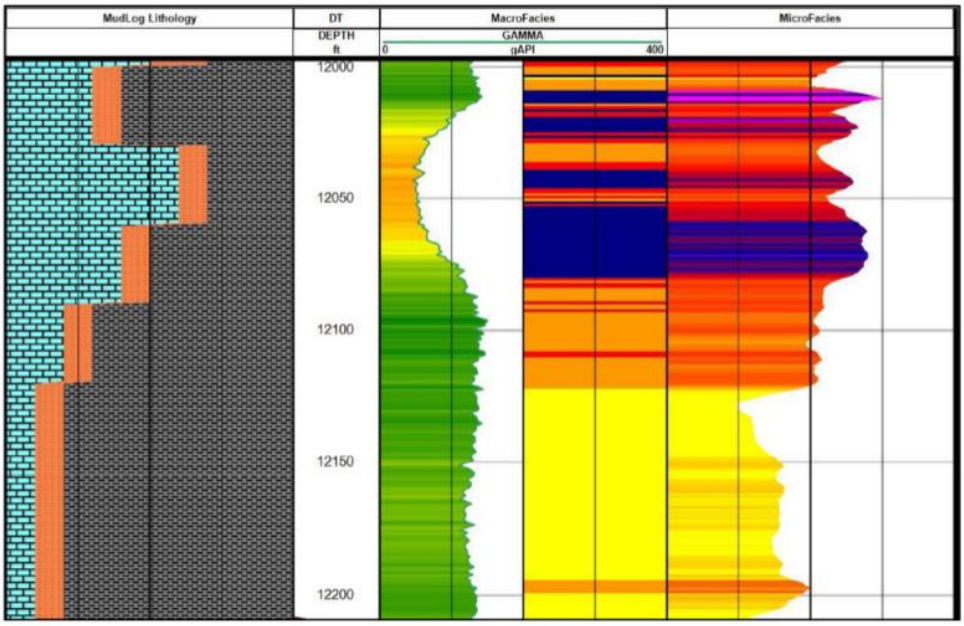


Figure 2.2: MSE (depicted as Macrofacies and Microfacies) as a Qualitative Indicator of UCS Trend (Logan, 2015)

The fourth (Macrofacies) and fifth (Microfacies) tracks in the log shown in Figure 2.2 are the outputs from Logan's MSE model. Each range in magnitude is called a "facies classification" where the MSE magnitude is interpreted to correspond to different rock facies. Based on how granular the classification range is, the facies classification can be considered a "micro" or "macro" facies classification. Examples of both are shown in Figure 2.2. Using the facies classification output, stage placement can be tailored to areas exhibiting the same MSE value in an attempt to target like rock (Logan, 2015).

Similar to Logan, Skinner, et al. (2016) investigated the validity of MSE as an indicator of rock hardness or strength and its usefulness in assisting engineered completions designs. They employed the same MSE term as Logan, Equation 5, and utilized the same surface measured data sources with some data quality screening and adjustment. Gamma Ray (GR) logs, geosteering final reports, and completions reports were analyzed along with the MSE data in an effort to link the drilling derived MSE term to the variability in the completions per stage. The results of their study however were inconclusive with regards to a direct correlation between the "like rock" strength derived from MSE and completions performance (Skinner, Van Domelen, & Grieser, 2016).

Ouenes, et al. (2017) with FracGeo published work on their Completions Optimization While Drilling (COWD) methodology, which incorporates the MSE calculation into a more complex and multifaceted workflow. Their methodology uses surface drilling data to estimate not only rock strength, or UCS, but also pore pressure and other geomechanical properties, such as Young's Modulus, Poisson's ratio, Shear Modulus, and Natural Fracture Index. These estimations are used to build a geomechanical model in real time to assist well steering decisions to ensure the well is placed in the best rock for a hydraulic fracturing treatment. It is then used to inform engineered completion designs and assist in offset well planning (Ouenes, Dirksen, Paryani, Rehman, & Bari, 2017). Figure 2.3 shows a workflow schematic illustrating the step by step process FracGeo has implemented, which begins with surface drilling data.

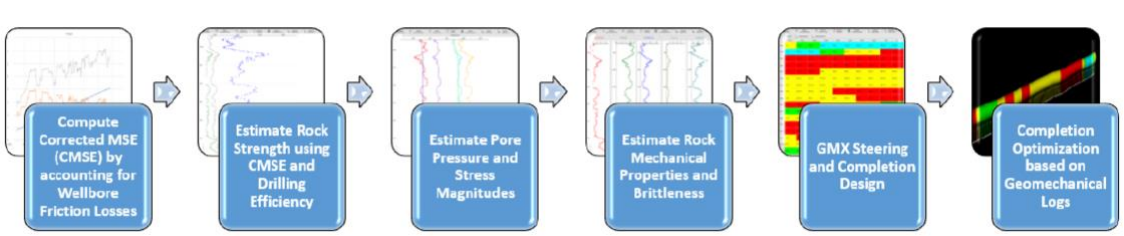


Figure 2.3: FracGeo COWD Workflow Schematic (Ouenes, Dirksen, Paryani, Rehman, & Bari, 2017)

MSE is again calculated using Teale's MSE equation. Recognizing that the parameters used to calculated MSE as measured at surface contain inaccuracies, a modified MSE term is proposed and defined as "corrected mechanical specific energy", or CMSE. The MSE term is adjusted using "very advanced drilling and wellbore mechanics" to estimate major frictional losses in the system that alter the torque and WOB values recorded at surface (Ouenes, Dirksen, Paryani, Rehman, & Bari, 2017). The resulting CMSE is more likely representative of the WOB and torque felt at the bit. This is in contrast to the approach presented by Logan, who does not account for frictional losses, how they change over time, and how they impact the value of MSE. Ouenes, et al. (2017) do not elaborate on the specific torque and drag and/or wellbore friction models employed in the presented methodology.

CCS is then estimated using the "classical rock strength criterion" which implies the use of a combination of relationships relating drilling data and rock properties found in the literature (Ouenes, Dirksen, Paryani, Rehman, & Bari, 2017). Warren (1984) and Kuru and Wojtanowsicz (1988) derived ROP prediction models using common drilling data for rollercone and PDC bits. The ROP models are based on a force balance between a PDC cutter and the formation rock. Additional work has enhanced the original ROP models,

including the introduction of bit design coefficients, estimated bit wear coefficients (Hareland & Rampersad, 1994), and chip hold down effects (Hareland & Hoberock, 1993). A detailed review of the literature covering the evolution of ROP models and their inversion for drilling and rock strength estimation applications is outside the scope of this work. However, they do not explicitly use MSE in their calculation, therefore it is unclear how MSE is used to derive CCS, as implied by Ouenes, et al. (2017).

These ROP models have been used to predict CCS with some success in combination with core derived lithology coefficients (Hareland & Nygaard, 2007). Therefore, some prior knowledge of the lithology is required to derive CCS using these methods, either from seismic or offset well logs. Thus, Ouenes, et al. (2017) imply a version of this work is employed, but is not explicitly described, to estimate CCS using the same corrected surface measured drilling parameters used to calculate MSE and some known lithological information.

In addition to the estimated CCS, an estimation of pore pressure (PP) and internal friction angle is required to calculate UCS, per Equation 8. Ouenes, et al. (2017) describe estimating PP using drilling data as well, but do not explicitly describe the method utilized in the COWD workflow. However, reference is made to the work by Majidi, et al. (2016) where the DE relationship is used to estimate PP, with prior knowledge of porosity in offset wells. While it may be implied that this is the method used, an investigation of the various methods for PP and internal friction angle using drilling data is outside the scope of this work. Ouenes, et al. (2017) acknowledge that if PP data is available via sonic and resistivity log interpretation, this can be substituted into the COWD workflow for the drilling data derived estimation.

With PP, internal friction angle, and CCS estimations, UCS is then assumed to be calculated using Equation 8. The estimated PP and UCS are then used to generate estimates of the minimum and maximum horizontal stresses and geomechanical properties. This information can be incorporated into geosteering and reservoir models in real time to update the trajectory plans for optimal well placement. When drilling is completed, the completions team has a geomechanical data set to use to design an engineered completion (Ouenes, Dirksen, Paryani, Rehman, & Bari, 2017).

2.3.3 Case Studies: Using MSE for Engineered Completions Design

Logan presents two case studies where the MSE log generated facies, or MSE value ranges, aligned with GR log derived heterogeneity in the well. These are presented as evidence that completion improvements could have been made through an engineered completion design based on MSE alone (Logan, 2015). The first case was a Wolfcamp well with a significant zone of natural fracturing identified by a suite of wireline logs obtained in the lateral. Plotting MSE along the lateral indicated an increase in MSE in this fractured area. Logan concluded that by using the presented methodology, stages would have been specifically placed in this fractured area, potentially improving the completion design (Logan, 2015). While targeting naturally fractured zones can be beneficial to production and the fractured area may be one definition of "like rock", this case does not necessarily provide evidence of an absolute UCS correlation with MSE. It does however identify a specific instance of a qualitative correlation within the generic "like rock" target for stage placement. Identifying the specific nature of the "like rock" section was not possible without the aid of additional data, which in this case was the wireline data obtained on this well.

The second case study analyzed the MSE value over the laterals of two offset wells that showed significant differences in production. The facies, or MSE log indicated more heterogeneity in the poorer performing lateral. Reviewing the MSE log within each stage, perforations that fell within the lowest MSE range were labeled "effectively treated" (Logan, 2015). This designation was based on the assumption that the lowest values of MSE corresponded with the weakest rock and that the treatment fluid will preferentially propagate a fracture in the weakest area of the formation. No completions data was used to aid in the analysis of each stage. Comparing the total number of "effectively treated" fractures between the two wells, Logan observed that the percentage increase between the two wells was similar to the percentage increase in production. He therefore concluded that MSE described the heterogeneity in the laterals accurately, and could have been used to target similar MSE values for each stage placement. This would have improved the completion and ultimately the production performance of the more heterogeneous well (Logan, 2015). However, production differences can be due to a variety of factors, not just from rock strength differences along the wellbore. These factors were not described or explored in Logan's evaluation and conclusions. The MSE changes may not have been due to rock strength changes, as demonstrated in the DE discussion in Section 2.2. Logan makes no mention of the well trajectory or if the lateral went out of zone at any time, which may also impact production potential. Therefore, his conclusions are interesting in the discussion of MSE and "like rock" relationships, but inconclusive in their definition of a direct correlation.

The study conducted by Skinner et al. (2016) began by comparing the calculated MSE along the laterals of three identically drilled wells that were subjected to geometric completions. Evaluating two stages within one well at the same vertical depth, variability

in the average MSE and completions energy (measured by treatment pressure, flowrate, fluid/proppant volumes, HHP, and instantaneous shut in pressure, or ISIP) was observed, indicating lateral heterogeneity in this well. Additionally, two consecutive stages were compared and showed little variability in average MSE (about 3,000 psi) with approximately 10 feet of vertical separation between the stages. However, the completion required different values of treatment rate and pressure, with only a slight difference in MSE. With only three wells of data, the study was not able to identify definitive trends in MSE and completion performance (Skinner, Van Domelen, & Grieser, 2016).

To expand the data set, Skinner, et al. (2016) observed two additional wells during drilling and completions operations. The completions were also performed in a geometric method. The first well had a relatively consistent MSE value over the length of the lateral, with a range between 0 and approximately 265,000 psi, and an average of approximately 80,000 psi and modal range of 70,000 to 78,000 psi. The completions operations per stage were also fairly consistent in their energy requirements, and showed similar ISIP and HHP requirements across all stages. The geology was a carbonate, and vertical pilot hole logs indicated consistent porosity within the target interval. Based on the comparison between completions and MSE data, the stable MSE appears to be influenced by the relative homogeneity along the lateral (Skinner, Van Domelen, & Grieser, 2016).

The second well contained more variability in the MSE value. The MSE had a range of 0 to approximately 400,000 psi, with an average of 85,000 psi and a modal range of 34,000 to 38,000 psi. Contrary to expectation, the completions were fairly consistent across all stages. They attribute the region of MSE peaks to the lateral going briefly out of zone into a dolomitic layer. Heterogeneity in the porosity was observed in the target zone as well via vertical pilot hole log data. Extrapolating this porosity out laterally, a negative

correlation was observed between MSE and porosity, where high MSEs were associated with lower porosities (Skinner, Van Domelen, & Grieser, 2016). Skinner, et al. (2016) note that in a carbonate reservoir, porosity may be an indication of rock hardness. While this appears to fit the expectation of MSE and rock hardness correlation, the completions did not indicate a correlation with MSE.

The methodology presented by Ouenes, et al. (2017) is more complex and dynamic than that proposed by Logan (2015) and investigated by Skinner, et al. (2016). They approach the problem from a multi-disciplinary view, combining all available data sources when possible and adapting the workflow to the most reliable data on hand. The key components however are surface drilling data, which is used to derive all subsequent geomechanical logs. If log data is not available to estimate pore pressure in the traditional methods, surface data is also used to estimate pore pressure. The result is a geomechanical log set that can be combined into a 3D earth model for well planning, completions design, and fracture modeling (Ouenes, Dirksen, Paryani, Rehman, & Bari, 2017).

To validate their approach, the geomechanical information estimated using surface drilling data was compared to that from log data obtained on a Wolfcamp well in the Permian Basin. The log data was interpreted using a 3D continuous fracture model (CFM) method to calculate a natural fracture index (NFI). The CFM used 11 offset wells with conventional log data to generate 3D mapped natural fracture proxies, and validated these against 2 additional offset wells with image logs. An NFI was then generated from the model results. The log data was also used to derive logs of shear modulus and Young's modulus along the length of the well. The log derived geomechanical data was compared to the NFI, Young's modulus, and shear modulus derived using the surface drilling data

methodology, the results of which are shown in Figure 2.4 (Ouenes, Dirksen, Paryani, Rehman, & Bari, 2017).

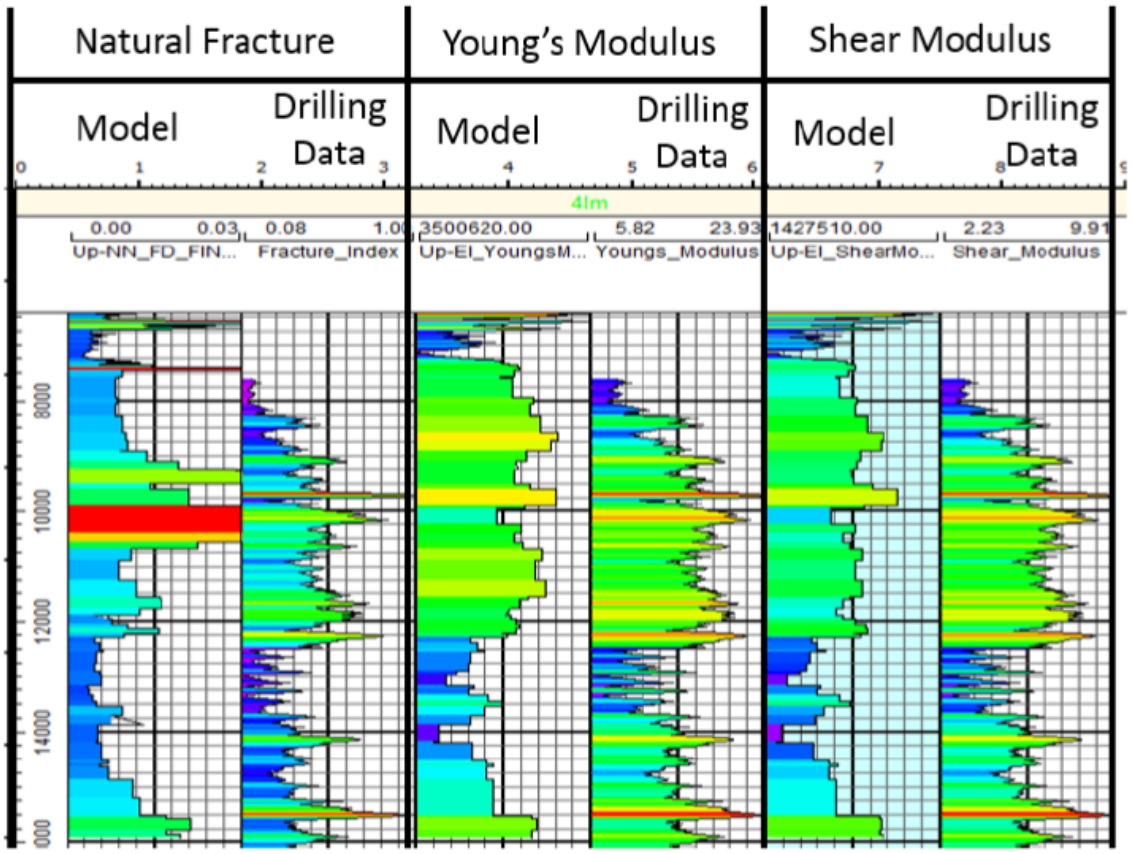


Figure 2.4: CFM Model Derived and Geomechanical Derived Geomechanical Parameters (Ouenes, Dirksen, Paryani, Rehman, & Bari, 2017)

The comparisons in Figures 2.4 show relative similarities between the log derived and drilling data derived geomechanical data. We see high ranges and low ranges of each property log occurring at similar depths, indicating a good agreement between the outputs from both methodologies.

The log results from the COWD method were also evaluated in a well in the Vaca Muerta formation in Argentina. The calculated NFI was compared to log derived conductive and resistive fracture density along the lateral of the well. The results of this comparison are presented in Figure 2.5 and 2.6.

Figure 2.5: Drilling Data Derived FI (left) and FMI Derived Conductive Fracture Locations (right) (Ouenes, Dirksen, Paryani, Rehman, & Bari, 2017)

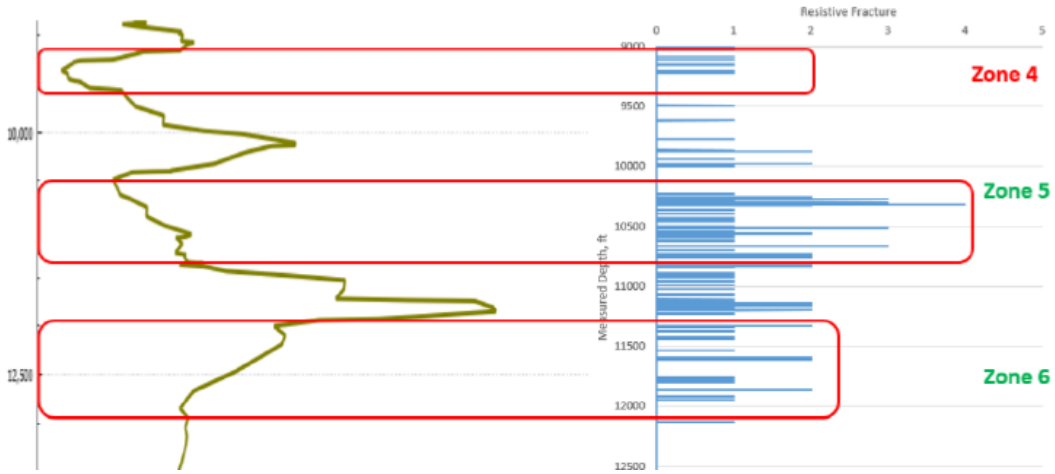


Figure 2.6: Drilling Data Derived FI (left) and FMI Derived Resistive Fracture Locations (right) (Ouenes, Dirksen, Paryani, Rehman, & Bari, 2017)

As shown in Figure 2.5, the peaks in drilling data derived NFI correspond with depths where log interpreted conductive fracture densities are highest. Conversely, the minimums in the NFI trend appear to associate with areas of high resistive fracture density, as shown in Figure 2.6, although not as significantly. The use of NFI as a qualitative indicator for conductive natural fractures seems reasonable in this example, which is useful for engineered completions design considerations. However, more work is required to correlate drilling data with the resistive fracture densities.

In addition to fracture densities, the density of recorded microseismic events in this well was compared to the shear modulus calculated from drilling data (Ouenes, Dirksen, Paryani, Rehman, & Bari, 2017). These results are shown in Figure 2.7.

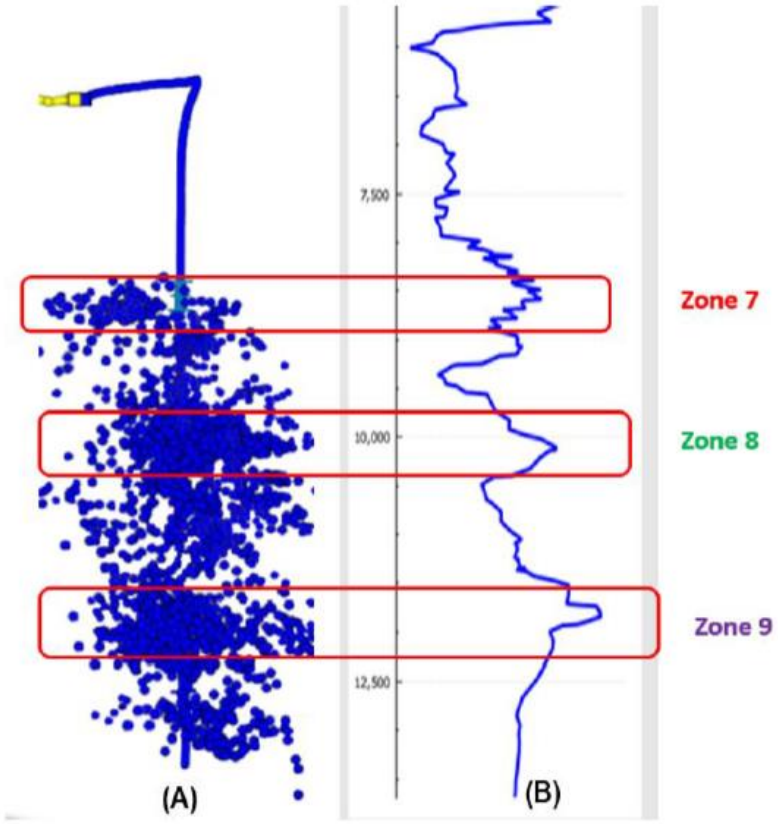


Figure 2.7: Microseismic Events (left) and Drilling Data Derived Shear Modulus (right) (Ouenes, Dirksen, Paryani, Rehman, & Bari, 2017)

The highest densities of microseismic events are seen at depths where peaks of shear modulus were calculated, and lowest when relative troughs in the shear modulus occur. This is another potential validation of the drilling data derived geomechanical properties generated in the COWD workflow (Ouenes, Dirksen, Paryani, Rehman, & Bari, 2017).

2.3.4 Discussion on Limitations of Current Methodologies

Logan acknowledges that MSE changes with the DE state of the system, however he argues that over the depth of interest from an engineered completions perspective DE

can be assumed constant (Logan, 2015). He does not identify efficiency fluctuations as a basis for data cleaning or MSE adjustment. For example, as part of the described data cleaning process in Logan's methodology, a low ROP limit is set to 3 feet/hour to prevent MSE spikes due to ROP as a denominator in the MSE equation (Logan, 2015). Based on the work described in Section 2.2, at low ROP, either at the beginning or end of a drilling period, the MSE term is likely inaccurate for the purposes of describing rock strength. Low ROP may occur frequently during the brief ramp up and ramp down of parameters in a drilling period, where bit engagement is moving in or out of the efficient drilling state. Setting a lower ROP limit just above zero does not completely eliminate these periods. No effort was described that seeks to remove or otherwise flag these brief periods, or other periods of potential inefficiency, as areas of inaccuracy when interpreting rock strength from MSE. For the case of ramp up and ramp down parameters, the depth over which these are captured was likely deemed negligible compared to the average stage length, and thus their impact is ignored.

In addition to potential inefficiencies, the physical state of the system changes significantly between slide and rotary drilling. This change results in different average MSE values between the two drilling states under reasonably efficient conditions. He does not consider the nature of the change in the system or its impact on DE between slide and rotary drilling to account for this difference. Instead, a parameter "offset" is applied to the differential pressure reading to shift the MSE curve during slide drilling into a comparable range with MSE values observed during rotary drilling. This shift allows the MSE log to be relatively continuous over both drilling states (Logan, 2015). Under the assumption that DE is constant, in both slide and rotary drilling states, MSE will change only with respect

to rock strength changes and therefore indicate heterogeneity along the wellbore, which is the foundation of Logan's methodology.

Logan's methodology uses a simplified MSE approach and potentially subjective interpretation of the data. The subjectivity that exists not only slows down the engineered completion workflow as a person must interpret the MSE and recommend a completion design, it also brings doubt into the validity of the MSE to rock strength correlation itself as consistency may be difficult to achieve. The methodology Logan presents may require more complexity than he suggests in his description. This complexity is explored in Chapter 4.

Contrary to Logan, the study conducted by Skinner, et al. (2016) concluded that MSE may be useful in evaluating drilling performance and potential lithology changes, however it is not the sole metric that should be used when making engineered completions decisions. Skinner, et al. (2016) investigated the approach presented by Logan by tying the MSE log to completions data. As no direct and consistent correlations between completion performance and MSE were found, they recommended that MSE be combined with other data sources for a better-informed view of what is happening downhole, and how the rock properties of interest to engineered completions may be derived (Skinner, Van Domelen, & Grieser, 2016).

The methodology presented by Ouenes, et al. (2017) and marketed by FracGeo appears to be the most promising MSE based engineered completions approach presented in the literature. The combination of a variety of drilling models, including MSE, torque and drag, ROP inversion, and a variety of empirically derived geomechanical correlations, results in a more data driven result than simply using Teale's MSE as a proxy for rock heterogeneity. A drawback to this approach is the inclusion of a high quantity of estimated

parameters to calculate each step in the log generating process, introducing more and more uncertainty with each layer of calculation. The COWD workflow however is built to allow for traditional data acquisition methods to be substituted for the drilling data derived estimations to enhance the reliability of the output logs when available (Ouenes, Dirksen, Paryani, Rehman, & Bari, 2017).

Chapter 3: Methodology

The research presented in this thesis was based on the work presented in the literature and described in Chapter 2 to derive rock strength changes from commonly available drilling data. The authors and companies who have previously worked in this field showed promise in a variety of areas for the application and use of MSE to indicate useful drilling and completions information. The simplicity implied is an area this work investigates and thus many of the methods used were based on the work already completed in this area. Additional methods were investigated for improving the currently used models and methods.

3.1 Drilling Data and Data Cleaning

Drilling data sets from 10 wells drilled on one pad location were used as the basis for this work. The wells investigated were drilled in the Spraberry Trend of the Permian Basin in West Texas. The trajectories of the 10 wells are shown in Figures 3.1, 3.2, and 3.3 with varying points of view. The wells are labeled 0 through 9.

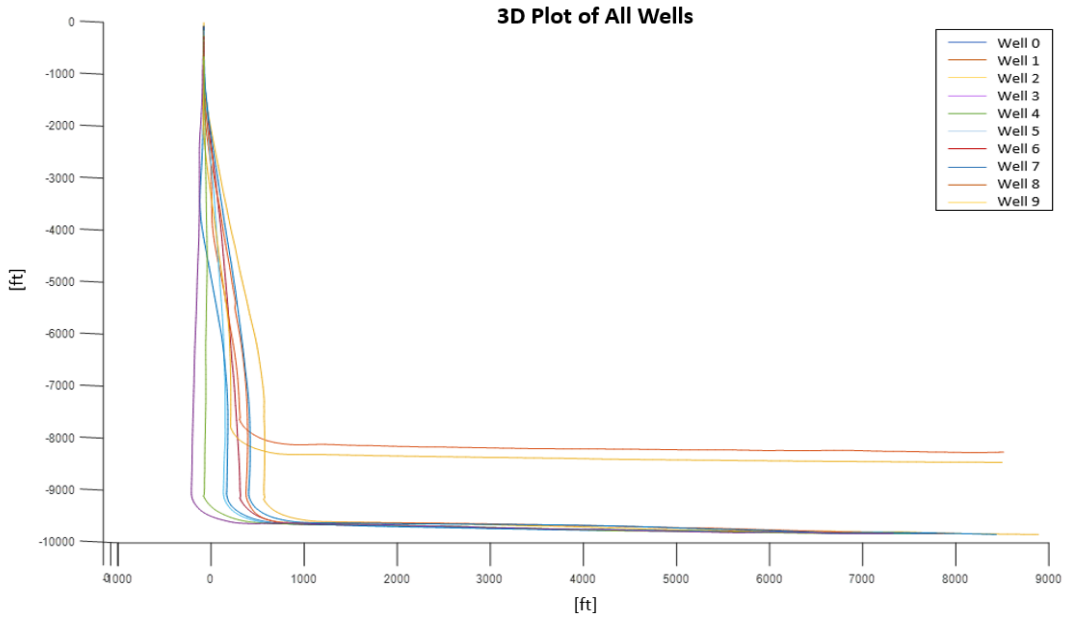


Figure 3.1: 3D Well Trajectories of Wells Analyzed, Side View

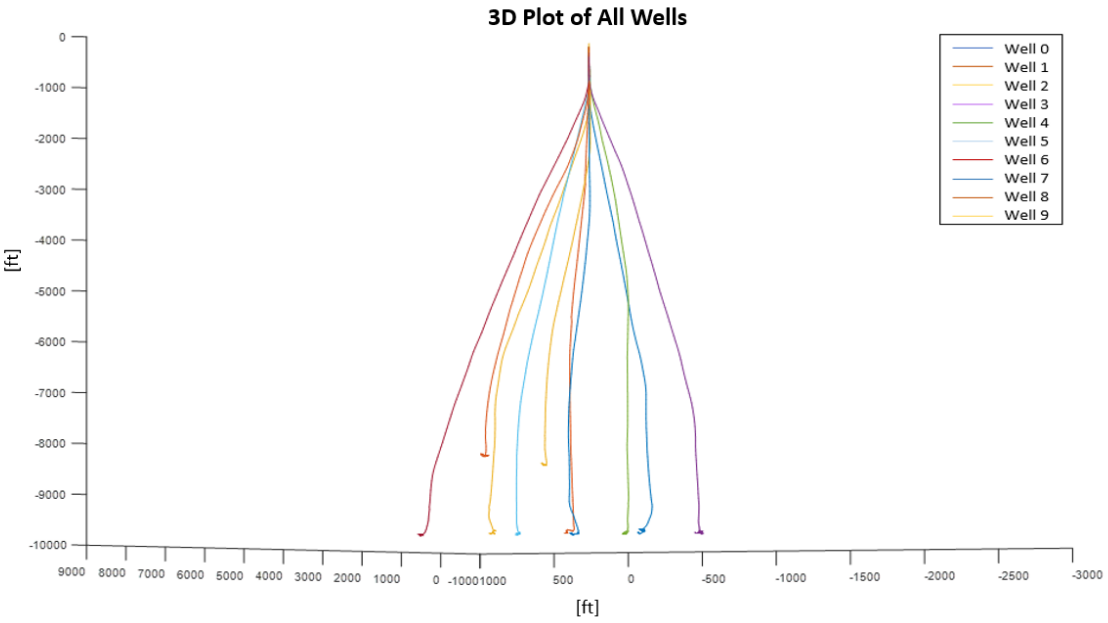


Figure 3.2: 3D Well Trajectories of Wells Analyzed, Heel-side View

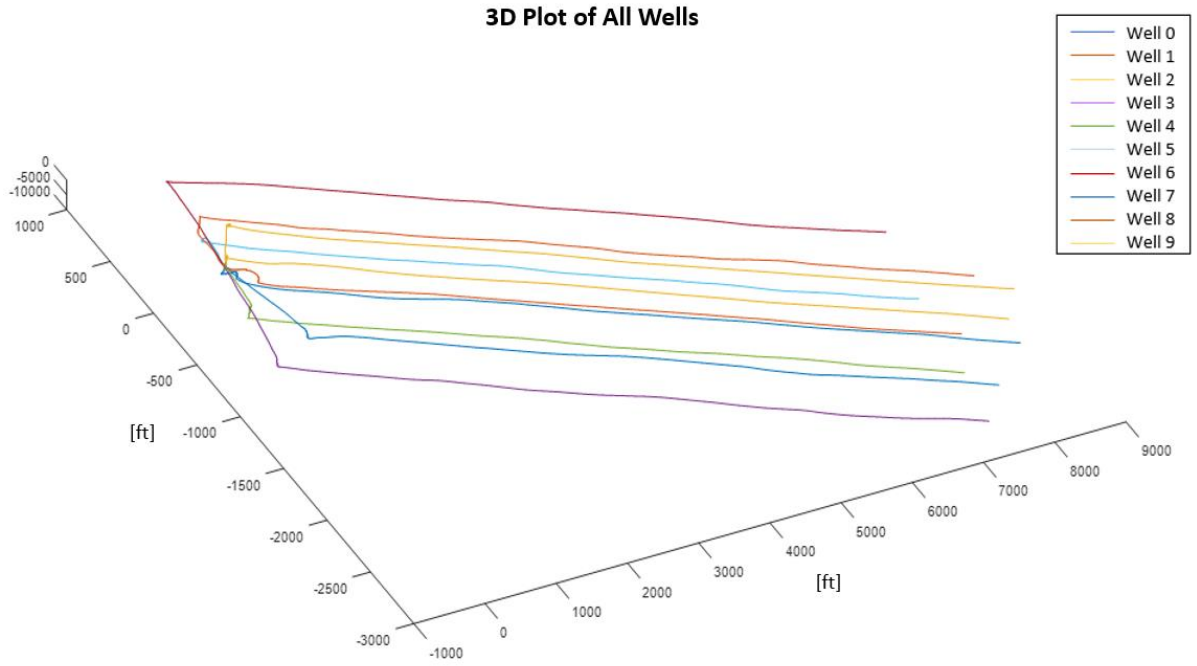


Figure 3.3: 3D Well Trajectories of Wells Analyzed, Top View

All 10 wells used a downhole mud motor, and drilled horizontal laterals that landed at approximately 9700 feet (ft) measured depth (MD), with the exception of two wells. Well 8 lateral was landed at approximately 7900 ft MD and Well 9 lateral was landed at approximately 8575 ft MD, shown in Figure 3.2. Each lateral was 7000 to 8000 ft in length.

The data used in this analysis originated in various formats and from various sources. Therefore, the data for each well was aggregated, manipulated, and cleaned in preparation for data analysis. The overall data handling and cleaning process for each well is summarized as follows:

- Combine raw data into master data set (per well)
- Calculate TVD from survey data
- Map offset UCS data to TVD in master data set

- Clean master data set
- Calculate MSE and other analytical metrics

The drilling data was provided in a raw, time-based format, taken from a real time continuous drilling data collection software system and measured every 10 seconds. Time was converted from a calendar-based time stamp format to cumulative time in hours. The data was an aggregation of surface measured parameters, such as WOB and RPM. The drilling data sets for each well were supplemented by geosteering survey data sets, BHA and drill string design data for each drilled section, and data extracted from a proprietary drilling dysfunction evaluation tool owned by the company which drilled and operates the wells. In order to compare drilling data to rock strength, the sonic derived UCS data from three offset wells was provided, as well as the sonic derived UCS data taken on one of the 10 wells investigated, Well 8.

A Matlab algorithm was created to extract the data streams of interest from each of the individual data sets and combine them into a unique data set for each well for further analysis. The data streams, or headers, extracted from each subset of data are shown in Table 3.1.

Table 3.1: Data Headers Extracted from Raw Data Files

Raw Drilling Data	BHA and Drill String Data	Dysfunction Evaluation Data
Hole Depth (MD)	BHA Number	Depth (MD)
Weight on Bit (WOB)	Bit Size	Bit Balling Belief
Bit RPM	Revolutions per Gallon	Bit Bounce Belief
Differential Pressure (ΔP)	Max Torque	Stick Slip Belief
Rate of Penetration (ROP)	Max Differential Pressure	Whirl Belief
Motor RPM	Start Depth	Other Drilling Dysfunction Belief
Block Height	End Depth	No Drilling Dysfunction Belief
Rotary RPM	Nozzle Total Flow Area [in^2]	
Total Pump Output (Q)	Mud Weight [ppg]	
Bit Depth		
Rotary Torque		
Inclination		
Azimuth		

The BHA and drill string design data was provided based on the start and end depths of each BHA run. A Matlab program was written to match the MD references of the data sets to the raw drilling data. The associated data from the additional data sets was then pulled into the raw drilling data set as a new set of headers. This program is presented in Appendix B. The program identifies the closest match in MD within the drilling data to the reported start and end depths of each BHA run. With the MD range identified, the associated design information is then applied to all lines of data within that depth range.

The UCS data was provided based on true vertical depth (TVD), obtained using wireline logging tools in a vertical pilot hole section of the target interval. At the end of the

vertical section of the well, an additional amount of vertical hole, or pilot hole, was drilled through the formation target interval. Wireline logging tools were then deployed and took various measurements of the formation over a select TVD interval for analysis of the formation properties. Sonic logs were obtained and converted into UCS data points along the vertically logged wellbore. This data was provided in units of pounds per square inch (psi) and already converted from sonic velocity into UCS values. The conversion from sonic to UCS is typically done using empirically derived constants from laboratory core experiments. In this case, the company providing the data for the project did not provide the sonic velocity data or the specific conversion constants used. Additionally, the nature of the measurement means the resulting UCS values have a limited resolution. The tool itself is a fixed length with source and sensor separated by a fixed distance. The resulting measurement is therefore an averaged result of the sonic travel time through the lithology or lithologies present within the distance traveled. This is evident in the smoothness of the UCS trend. Therefore, if a sudden change occurs, it is not necessarily captured as a stepwise change in UCS value. This is important when comparing drilling data and the sonic derived UCS value, as there is some inherent error in the sonic measurement conversion to UCS.

To combine the UCS data provided with each well's master data set, the TVD references between the two data sets much be matched. TVD for the drilling data set was calculated using the Average Angle Method (Walstrom, Harvey, & Eddy, 1972). Using the inclination (Incl) from the real time survey data, obtained from the raw drilling data set, and the measured depth for each data point, Equation 11 was used to calculated TVD.

Equation 11

$$TVD = \sum_{1}^{i} (MD_i - MD_{i-1}) * \cos\left(\frac{Incl_i - Incl_{i-1}}{2}\right)$$

The associated Easterly (CoordEast) and Northerly (CoordNorth) directional offsets were calculated using the average inclination and azimuth (Azim), calculated between every two consecutive data points of MD. These values and the length of the MD between the two points were used in Equation 12 and 13 to calculate the directional coordinates, assuming the starting point is (0,0) of a coordinate system.

Equation 12

$$CoordEast = \sum_{1}^{i} (MD_i - MD_{i-1}) * \sin\left(\frac{Incl_i - Incl_{i-1}}{2}\right) * \sin\left(\frac{Azim_i - Azim_{i-1}}{2}\right)$$

Equation 13

$$CoordNorth = \sum_{1}^{i} (MD_i - MD_{i-1}) * \sin\left(\frac{Incl_i - Incl_{i-1}}{2}\right) * \cos\left(\frac{Azim_i - Azim_{i-1}}{2}\right)$$

Using Equations 11, 12, and 13 above, the TVD and spatial coordinates of the bottom hole were found for each time stamp. These could then be mapped in 3D to generate the wellbore trajectories shown in Figures 3.1, 3.2, and 3.3.

With the TVD known, the TVD references of the UCS data were matched to the TVD of the drilling data, mapping the UCS data to the master drilling data set. The UCS data was taken on a 0.5 ft TVD scale, while the drilling data was time based and thus more granular in its capture of TVD. Therefore, every TVD point in the drilling data was rounded

to the nearest 0.5 ft, and the associated UCS value was assigned to every instance of its corresponding TVD in the drilling data. This results in a UCS stair step affect when plotted with depth, particularly as inclination increases due to longer time intervals within the same TVD at a 0.5 ft scale.

With the master data sets compiled for each well, the data was then cleaned to remove time intervals where the bit was not engaging with the rock to progress drilling. A Matlab program was written to execute this step in the data manipulation process, and is presented in Appendix F. The data cleaning methodology utilizes a generic framework of code that was then applied to different conditions within specific data headers or variables. The generic code structure began by taking the difference between each data point within a specific variable to find the derivative of that variable. In some cases, a derivative was not necessary and the cleaning code could be applied directly to the variable. Then a conditional find function was applied to the derivative or variable to identify the location (or index) of unwanted data. The index of each unwanted data point within the variable of interested was then applied to every header in the master data set, and the row of data corresponding to that index was removed. The variables and their conditional requirements for cleaning the data are summarized in Table 3.2.

Table 3.2: Data Cleaning Methodology

Variable	Remove if…
Block Height	upward movement of traveling block
ROP	less than or equal to zero
Differential Pressure (ΔP)	less than or equal to zero
Hole Depth (MD)	no change
Bit Depth	no change or negative change (moving up)
Total RPM (Mud Motor + Rotary)	less than or equal to zero
WOB	Less than or equal to zero
Total Pump Output (Q)	Greater than or equal to 1000 gal/min (remove spikes)

The data cleaning methodology presented in Table 3.2 removes time intervals over which no forward drilling was taking place. This is accomplished through the removal of data where the traveling block was moving upwards in order to make a connection of a new stand of drill pipe, the removal of instances where the bit was not on bottom or moving backwards (i.e. tripping out of the hole), or removal of periods when the measured depth was not progressing. To ensure quality in the calculation of MSE, instances where the drilling parameters were not indicative of forward drilling were also removed. This includes when ROP, WOB, RPM and ΔP were negative or equal to zero. As it is physically impossible for these parameters to be negative, removing the negative values (if any occur) removes bad data that should not be used in the MSE calculation. A ceiling was placed on the mud flow rate value of 1000 gal/min to remove erroneous spikes in the data. This is an adjustable value, and can be set to the maximum flowrate rating of the pumps used. For the

data sets used, 1000 gal/min was sufficient to eliminate unrealistically high values of flow rate.

With the data aggregated and cleaned for each well, the master data sets could then be used to calculated MSE and other metrics for analysis. The cleaning methodology presented here does not remove instances of slide drilling so that these can be captured and analyzed using the methods described in the following sections. However, it is important to distinguish between slide and rotary drilling as the resulting values of MSE are significantly changed between the two drilling states. Any data analysis carried out on a drilling data set should therefore review the two drilling states separately. Because slide drilling is associated with no rotation of the drill string, one would assume that isolating the instances where rotary RPM is zero would capture slide drilling in the data. However, the rotary RPM is not always precisely zero while slide drilling. This is due to some surface movement at the sensor as the string is progressing forward, as well as minimal rotation intentionally induced by the driller, called "rocking", to break up the friction the drill string experiences under sliding conditions. Therefore, after a review of the data under known sliding conditions, a minimum threshold for rotary RPM was set to 60 RPM, whereby if it fell below this value it was assumed to be in a slide drilling or equivalent state.

3.2 MSE CALCULATIONS

To calculate MSE, the modified Teale's equation presented in Equation 5 was used to account for the presence of a mud motor. The original equation as defined by Teale has primarily remained unchanged in the literature, with the exception of Armenta's hydraulic contribution. However, many of the companies using MSE as a rock strength indicator do not apply Armenta's additional term, or do not explicitly describe their use of it in their

published work. They may be accounting for hydraulic impacts on the MSE in other ways, but do not reveal their methods in detail for proprietary reasons. To simplify the use of MSE in this research, the hydraulic term was ignored and Teale's equation was used with the simple changes to reflect the contribution of a mud motor. Torque applied at the bit is thus estimated as the ratio of the mud motor design parameters of maximum torque and maximum differential pressure, multiplied by the differential pressure.

A Matlab program was written to calculate MSE at each time stamp in the drilling data, and is presented in Appendix C. The input variables were WOB, ROP, Rotary RPM, Bit Size (diameter), Total Pump Output (or flow rate, Q), ΔP, the revolutions per gal rating of the mud motor, and the max ΔP and torque rating of the mud motor. The total bit RPM was calculated per Equation 5, combining the Rotary RPM and calculated mud motor RPM. A mud motor RPM header was included in the provided raw drilling data sets, however a mud motor RPM was calculated separately using the mud flowrate and revolutions per gallon rating of the mud motor, and used in calculating the MSE. The units of each variable were adjusted to produce an MSE value in units of psi.

3.3 Manual Data Analysis Approach

The data analysis presented in this work began by investigating the reliability of Logan's methodology for using MSE as an indication of rock strength changes. Because Logan assumes DE is constant, any change in MSE would be due simply to a change in rock strength (Logan, 2015). The DE term however includes minimum levels of inherent inefficiency, as well as any changes in the inefficiency state of the system, both of which are reflected in the MSE term. The data analysis therefore explored the areas where MSE appeared stable and unstable, with the intention of identifying characteristic behaviors in

the stable data to indicate specifically a rock strength change, and behaviors in the unstable areas that could identify a change in DE.

MSE, UCS, and the parameters used to calculate MSE were analyzed by plotting and manually reviewing for characteristic trends and patterns to describe the efficiency state of the system. As the analysis progressed, the manual data analysis moved further towards machine learning methods for pattern recognition.

3.3.1 Trend Analysis

To evaluate trends in the MSE data that relate to changes in rock strength, MSE was plotted versus time and depth, both measured and true vertical. The MSE values were compared to the offset UCS data when possible to evaluate similarities between the two trends. To better visualize macro trends, MSE was smoothed using a moving average window of 2 minutes. The smoothed and the raw data for MSE were plotted and analyzed together. Data analysis took place at both a macro level, or zoomed out scale to hundreds or thousands of feet MD equivalent, and a micro level, zoomed in to tens of feet MD equivalent.

DE was calculated as the ratio of MSE to UCS and plotted for analysis. However, the UCS data provided was for a limited depth range. This is expected as logging data is not typically obtained over the entirety of the wellbore, and is focused on the potential reservoir sections of the well. The depth range, on a TVD basis, where offset UCS data was available was between 5700 and 11800 feet. The UCS data obtained on Well 8 was more limited and was within the range of 7500 to 8600 feet TVD. Figure 3.4 shows the four different sources of UCS data plotted together vs. TVD to visualize the variability in the data sets.

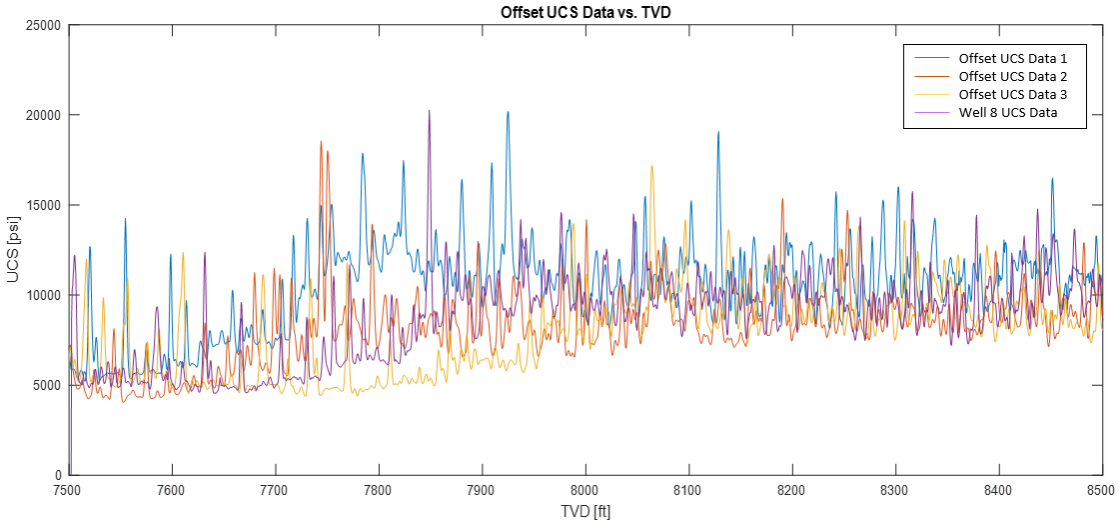


Figure 3.4: Offset and Well 8 UCS Data vs. TVD

Due to the variability and limited depth range of the UCS data available, DE was also variable and only available at these depth ranges. The minimum MSE method, Equation 7, was not used to calculate DE as this analysis aimed to distinguish precise behaviors related to UCS changes. Using the minimum or baseline MSE in this case would be the same as using a constant UCS as the numerator in the DE equation, with some fixed multiplier. Therefore, to test Logan's constant DE theory, calculating DE using the baseline MSE approach was not necessary. In cases where DE and UCS are changing together, quantitively isolating the magnitude of influence the UCS change has on the MSE term is unknown and difficult to calculate, and may induce a change in the baseline DE. Therefore, a direct UCS to MSE ratio was taken as the DE for this data analysis. However, the MSE term itself was used in the majority of the analysis to recognize areas of inefficiency instead of the DE term due to the variability in the UCS data provided. Therefore, the relative fluctuations in MSE, particularly in areas of relatively high MSE, were used as qualitative

indicators of inefficient drilling periods, similar to the real time approach for drilling optimization presented by Dupriest and Koederitz (2005) and discussed in Chapter 2.

Where direct UCS comparison was not possible, the analysis compared well to well data sets to view similarities in the MSE trends. If similarities in the trends were observed, this may indicate similar rock strengths laterally through the formation as it is highly unlikely that the same drilling inefficiencies would be introduced at the same depths in two different wells. Slight vertical offset may also be observed through comparison of MSE trends. At the micro level, the best data for analysis was in the shallowest sections as there is less frictional losses in the system and potential for noise in the data. Irrespective, a micro and macro assessment was made at all depths of each well.

In the process of reviewing the MSE data, it became apparent that the MSE term itself was not the only trend useful in visualizing relationships between MSE and rock strength change. Because the MSE behavior is dependent on the behavior of the input drilling parameters, the parameter data streams were also plotted versus time and depth and analyzed. The parameters were also smoothed using a 2-minute moving average, and were evaluated individually, as well as against each other. Dupriest and Koederitz (2005) describe linear relationships between parameters that can be used to indicate which region along the drill off curve the system is operating in, as discussed in Chapter 2. Therefore, cross plots of parameter combinations were generated and used to help isolate areas of potential high efficiency and low efficiency. These plots were generated in Spotfire, a data visualization and analytics software tool. Areas where similar efficiency states appeared to occur were manually analyzed in an attempt to identify characteristics trends to indicate UCS changes or system efficiency changes.

3.3.2 Defining Parameter Behavior

Analysis of multiple parameter trends taking place together was a process that was difficult for the human eye. Therefore, programmatic methods using Matlab were explored to assist in the data analysis and pattern recognition process. This began by taking the derivatives of each parameter and performing a similar visual analysis as described in the previous section. Parameter derivatives were cross plotted and compared to the MSE trend in an attempt to identify dynamic behaviors that could be automatically identified as UCS changes or other influences on MSE.

3.3.2.1 Manual Behavior Hypotheses for Slope Combinations

To enhance the efficiency of the data analysis, a group of programs were written in Matlab to classify behavior combinations and identify where these behaviors occurred along the wellbore. These programs are presented in Appendices L through U. The behavior analysis began by investigating the combination of slopes or derivatives of WOB, ROP, and torque. RPM was excluded from the behavior analysis as it tends to be relatively constant, with any change occurring almost instantaneously. Although it is not included in the parameter behavior definitions presented here, it is a significant feature of the broader drilling context as the level of RPM may influence when a founder point or ROP limiter may occur. The MSE slope was not used in the behavior trend analysis as the parameter trends themselves define the MSE trend. The slope of each parameter was found using two methods. The first method applied a point by point derivative of the 10 second data. The second method took an estimated slope over a bin or window of data. The analysis presented in Chapter 4 is based on a 1 minute bin, encompassing six consecutive data points. The slope was found by averaging the first two and last two data points, then taking the difference of the averages.

The initial parameter slope combinations analyzed were the linear relationships described by Dupriest and Koederitz (2005) associated with efficient behavior. These relationships are based on the assumption that UCS is constant, therefore if DE is constant and relatively efficient, they should occur when UCS is not changing. The relationships are summarized in Table 3.3, where the arrows denote the direction of the parameter slope and "c." means the parameter slope is constant.

Table 3.3: Initial Hypotheses for Constant UCS Parameter Behavior

UCS Constant
WOB ↑, ROP ↑, Torque ↑
WOB ↓, ROP ↓, Torque ↓
WOB c., ROP c., Torque c.

To expand the hypothesized UCS behaviors, additional parameter combinations were added to those presented in Table 3.3. These were based on an intuitive understanding of the bit reaction to an increase in UCS when the WOB is changing or held constant. As the rock strength increases, it will be harder for the bit to drill at the same ROP through the new rock, thus reducing ROP. If WOB is simultaneously increased, the increase in thrust will induce a counter increase in ROP. Therefore, the additional parameter slopes are based on the simplified case where the change in WOB negates the simultaneous effect due to the change in UCS. Each parameter combination was assigned an associated UCS change, generating the complete set of hypotheses shown in Table 3.4.

Table 3.4: Initial Hypotheses for Parameter Behavior Under Variable UCS Conditions

UCS Increasing	UCS Decreasing	UCS Constant
WOB ↑, ROP c., Torque ↑	WOB ↓, ROP c., Torque ↓	WOB ↑, ROP ↑, Torque ↑
WOB ↓, ROP ↓, Torque c.	WOB ↑, ROP ↑, Torque c.	WOB ↓, ROP ↓, Torque ↓
WOB c., ROP ↓, Torque ↑	WOB c., ROP ↑, Torque ↓	WOB c., ROP c., Torque c.

3.3.2.2 Characteristic Behavior Matrix and Statistical Behavior Assignment

The initial hypotheses presented in the previous section were assigned an associated UCS influence manually. With three parameters and three slope possibilities, the total possible parameter slope combinations were 27 in total. Instead of manually assigning a UCS slope, either positive, negative or constant, to each parameter slope combination not included above, a statistical approach was developed to assign the most likely UCS influence given the UCS changes observed when those combinations occurred in the given data set.

The statistical approach begins by defining the behavior of the parameters, again including WOB, ROP, and torque. To capture this description, a suite of Matlab programs were written which generate a characteristic behavior matrix (CBM) that captures different behavior traits and converts the overall description into a unique numerical label. The programs are presented in Appendices P, Q, R, S, and T. Each column of the CBM corresponds to a combination of behaviors that define a specific characteristic. For example, the first column defines the combination of parameter slopes. If at a given time step WOB is increasing, ROP is increasing, and Torque is increasing, a number corresponding to this specific sequence of behaviors would be assigned in the first column

of the CBM. The second and third columns of the CBM define the magnitude with which the parameters are moving. The classification for this behavior was simplified to either large or small movement in either direction (up or down). Additional columns could be added to the CBM to further define behaviors, however the analysis presented Chapter 4 uses only these two descriptors to analyze the data. Figure 3.5 shows a schematic representation of the algorithm, where the CBM is created at each timestep along the wellbore then used to assign a UCS prediction to each parameter behavior.

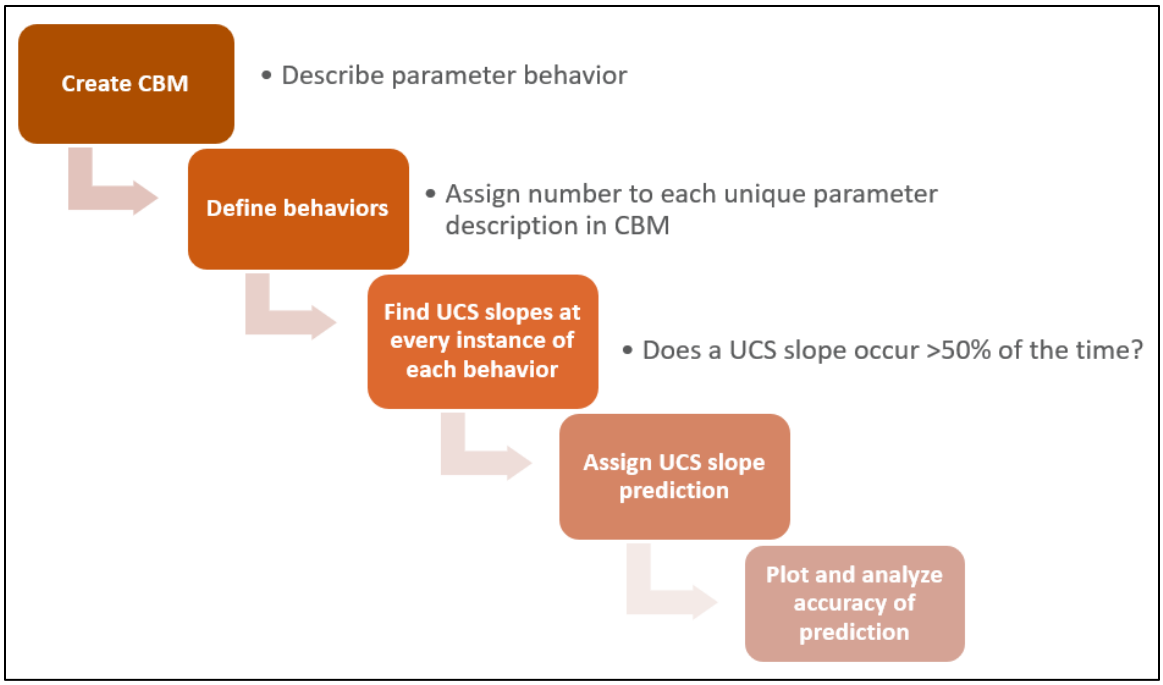


Figure 3.5: Workflow Schematic of CBM Implementation

To generate the first column of the CBM describing the combination of parameter slopes, the algorithm begins by taking the derivative of each parameter, as described in the previous section, and creating a corresponding vector in time that is the classification of the derivative sign or parameter slope. If positive, the classification is assigned as 1, and if

negative a -1. If the derivative is 0 then the classification is 0. Because the parameters are almost never constant from one time step to the next, a threshold was set to approximate constant behavior for each parameter. For ROP, if the derivative was between -2 and +2 ft/hr/10sec, ROP was classified as constant. Similarly, the thresholds for the derivatives of WOB and torque were set at -600 to 600 lbsf/10sec and -175 to 175 ft-lbs/10sec, respectively. These are adjustable inputs within the algorithm and can be used to change the granularity with which the CBM defines constant parameter behavior. Looking at a matrix of the three parameter slope classification vectors, the algorithm assigns a unique number identifying each specific combination of slope classifications at each time step, and places that number in the corresponding row of the CBM. Because there are 27 possible combinations of parameter slopes, the first column of the CBM will be assigned a number between 1 and 27.

In addition to the slope combinations, the algorithm defines the magnitude of change the parameter experiences between two consecutive time steps as the next column space within the CBM. The magnitude of change is classified as either large or small. The algorithm begins by defining how many parameters are "constant" at the given time step and placing that information in the second column in the CBM. This is assigned a 1, 2, 3, or 0 corresponding to the number of parameters that are constant. Next, the magnitude of the slope is defined as large, small or constant, assigning a 2, 1, or 0, respectively. These assignments are placed in three vectors, one representing each parameter. Using these vectors, the algorithm assigns a unique number to the third column of the CBM representing the combination of magnitude changes occurring. This classification step was broken down into groups depending on how many parameters are constant for simplification. For example, if there are no constant parameters at that time step, there are

8 possible combinations of magnitude of change. Therefore, the third column of the CBM will be assigned a number 1 through 8, while the second column of the CBM is assigned a 0. This method results in two columns of the CBM which together represent a unique numerical description of each possible combination of magnitudes of parameter change.

With the CBM generated, the algorithm finds all unique numerical combinations within the 3 columns and assigns a number to each. Thus, the CBM is converted into a numerical identification system which describes unique parameter behaviors along the wellbore. This numerical identification is added as a header to the master data set so that it can be plotted to identify where the behaviors occur in time and along the wellbore.

To assign a UCS prediction to parameter behavior defined in the CBM, the algorithm compares the CBM output to the slope of the UCS data provided. In the analysis presented in Chapter 4, the UCS data from Well 8 was compared to the CBM generated from the Well 8 well data. The accuracy between the UCS and drilling data is paramount for this analysis due the specific nature of the comparison being made. Therefore, the Well 8 UCS data was considered the best representation of what the bit was experiencing in this drilling data set. The alternative was to compare Well 8 UCS data to drilling data from another well, or compare UCS from an offset well to any of the wells in the study. Both of these comparisons would require lateral extrapolation of the UCS measurement between the wells and an assumption that there is minimal lateral heterogeneity in the formation. Because the basis of this work was to test the ability of the surface measured drilling parameters to detect lateral UCS heterogeneity, assuming lateral heterogeneity was negligible over the distance between two wells was not preferred. However, some extrapolation was required using the Well 8 data as the UCS data was acquired in a vertical pilot hole that primarily covered the curve and lateral sections of the Well 8 wellbore.

Therefore, the Well 8 curve section was isolated for analysis and the TVD based UCS data was mapped along the measured depth of the curve.

To define the UCS behavior, the derivative of the UCS was taken on a point by point basis and a slope classification applied, similar to the parameter slope classification previously described. A threshold was set to approximate constant UCS as well, and was set as + or – 50 psi/10sec in this analysis. The algorithm then uses a statistical approach to assign a "predicted" UCS slope to each parameter behavior as defined by the CBM, summarized in Figure 3.5. Each unique parameter behavior defined in the CBM is evaluated separately. The time steps or indices of all instances of a specific behavior are found, and the corresponding UCS slopes at those indices are extracted into a sub data set for evaluation. If one UCS slope class dominates the sub data set (i.e. occurs more than 50% of the time), that slope is assigned as the "predicted" UCS slope for that behavior. Thus, a UCS slope is "predicted" using that behavior description. If there is no dominant UCS slope within the sub data set, the behavior is labeled as "inconclusive." Where inconclusive data behavior occurs, either the current description of the parameter behavior is not detailed enough to distinguish it from other behaviors to indicate a specific UCS influence, or some inefficiency in the system is dominating the parameter behavior and masking any UCS change. If the latter is the accurate assessment, this would be an instance of "unreliable" MSE behavior when attempting to use MSE as a proxy for UCS. The results of this analysis are presented in Chapter 4.

3.4 MACHINE LEARNING APPROACH

A machine learning algorithm, specifically random forest, was investigated as an alternative approach to identify and characterize parameter behavior that can be tied to a

specific UCS trend. The random forest algorithm was chosen as it is fairly simple to use and implement within Python, and tends to result in a generalized output once the parameters of the model are optimized, reducing the potential for overfitting in the prediction model results. The study did not explore neural networks or other machine learning methods, and therefore does not present the random forest as the best machine learning approach for this application. Future work may benefit from a comparison of multiple machine learning methods to assess the benefits and disadvantages of each in the context of this problem.

3.4.1 Definition of Random Forest

A random forest is a commonly used supervised machine learning algorithm. It is based on the "bootstrap aggregating" or "bagging" method where multiple decision trees are combined into a "forest" to improve accuracy and reduce overfitting in the resulting prediction model (Donges, 2018). Supervised learning means that the algorithm is provided a predefined set of "features" and explicitly given the prediction answer, or "label", in the learning process. A decision tree is a series of binary questions (ex. true or false) that repeatedly split the data to get less and less error in the resulting prediction. The tree is essentially a map through a data set of features that leads to a known result, or label. A random forest is the aggregated result of multiple decision trees generated through supervised learning (Ronaghan, 2018).

Overfitting is a common problem in machine learning methods and is commonly observed in decision trees (Donges, 2018). Overfitting is when the prediction model generated is too precise in its description of the data within the given data set from which the model was built (or learned from). This results in very good agreement or accuracy of

prediction within the learned data set, but poor accuracy when the predictions are applied to a new or test data set. The bagging method utilized by the random forest algorithm combines the results of multiple, independent decision trees, creating more randomness in the overall prediction. Therefore, the tendency to overfit is reduced when compared to a single decision tree (Donges, 2018).

To generate each tree, the random forest algorithm randomly splits a data set into multiple subsets and generates decision trees on each subset of data. To further improve the randomness in the generation of each tree, each node or question where the data is split uses a randomly selected subset of features (Donges, 2018). The algorithm assesses each feature in the subset, picking the feature criteria where the best reduction in prediction error is gained at that node. The data is split into two branches based on that criteria and the random selection of features is repeated in each descending node. The tree grows until a maximum predictive accuracy associated with each label is achieved, creating an end node, or "leaf" where splitting or branching no longer occurs and a prediction is defined (Ronaghan, 2018). The aggregation of all trees produced by the algorithm is called the random forest. For a categorical prediction, the mode classification of the common predictive pathways from all of the trees in the forest is the prediction taken when given a new set of data. In a regression model where the predictions are continuous, an average is taken and used as the prediction result (Ronaghan, 2018).

3.4.2 Random Forest Algorithm Structure

A series of programs written in Python and Matlab were created to execute a random forest algorithm and generate a model to predict the UCS slope using predefined features in the drilling data. These programs are presented in Appendices V though Z. The

Well 8 drilling data and Well 8 UCS data sets were used to train and test the model. Again, Well 8 was chosen because the UCS log data was obtained on the same well as the drilling data.

To implement the random forest algorithm, the Random Forest Classifier function from the Scikit-Learn (sklearn) free software library was imported into the code. Sklearn and other Python software libraries (ex. Pandas, Numpy, Seaborn, Scipy, etc.) are pre-written Python algorithms that can be imported for free and used as functions in a new Python program. These libraries are useful tools when writing a variety of different Python programs, and make implementing a random forest and other machine learning algorithms fairly straight forward. Because the Random Forest Classifier is a pre-written algorithm that can be called upon once imported into the program, the required lines of code are condensed to those which define the inputs, outputs, and hyperparameters (or algorithm settings) and then a single line to call out the Random Forest Classifier algorithm. The Python code written for this study is presented in Appendix Y.

3.4.3 Random Forest Implementation

A group of Matlab and Python programs were written to execute the required tasks to manipulate the data into a format to be used in the Random Forest Classifier function. These programs are presented in Appendices V, W, and X. The algorithm trims the Well 8 master data set, which includes the Well 8 UCS data mapped to TVD, to only include the MD over which the UCS data was available along the curve section of the wellbore. The algorithm then uses the data to create vectors which describe the features and labels to be used in generating the random forest. The implementation of these programs which yielded the results discussed in Chapter 4 is described here.

3.4.3.1 Creating Training and Test Data Sets

When generating a random forest with a given data set, a portion of the data is set aside as the "training" set and the remaining data is designated as the "test" set. The training set is used to build the prediction model, which is then applied to the test set to evaluate the accuracy of the prediction model within the given data set. The random forest implementation in this study investigated two ways of splitting the data. The first was to randomly select the train and test data using the built-in capability within the sklearn random forest algorithm. The algorithm randomly selected 75% of the data as the training set, and used the remaining 25% as the test set. The percentages chosen are typical of a random forest implementation, but can be modified if the user so chooses. The second method was to split the data in half based on the measured depth. This was done using a Matlab program, found in Appendix W. After splitting the data, the second half of the curve section was used as the training set and the first half of the curve section as the test set. Advantages and disadvantages of both splitting methods are discussed in Chapter 4.

3.4.3.2 Defining Features and Labels

After creating the training and test data sets, the features were defined for use in the random forest algorithm. The features were created for both train and test data sets in an identical manner. Similar to the initial parameter behavior analysis, the random forest features only describe the ROP, WOB, and torque parameters within the data set. Again, RPM was not included in the evaluation because it was generally constant, with an occasional and relatively instantaneous step change to a new RPM setting. The study focused on parameter trend behavior, starting with the simple point by point derivative of the smoothed parameter data streams. As previously noted, the raw parameter data was smoothed using a two-minute moving average and the derivative of the smooth data was

taken between each consecutive point. The derivative was then converted into a classification, designated by a number ranging from -11 to 11. The parameter descriptions in the previous section used a simple positive negative slope classification, then defined a second classification to describe large or small movement. These two descriptions of behavior were combined into a more granular numerical classification of the slope, where -11 is the highest negative slope and +11 is the highest positive slope. The parameter derivative values were normalized to a -11 to +11 scale and then classified. If the normalized derivative fell between 0 and 1, a class of 0 was applied. If it fell between 1 and 2, a class of 1 was applied, and so on. Anything larger than 11 was applied class 11, and similarly in the negative direction. The derivative of each parameter was normalized based on the magnitude of the values. For example, WOB was captured in units of 1000 lbs (i.e. klbs), therefore the derivative was typically in the range of 0.1 to 0.9 klbs/10sec. Thus, the derivative was multiplied by 10 to normalize the values for classification. Like the previous analysis, a unique threshold could be set to approximate constant behavior for each trend beyond the 0 to 1 classification range. For example, the WOB threshold was set to 200 lbs. Therefore, any derivative value that fell within the 0 to 2 range was assigned a class of 0, defining the class 1 values as equivalent constant behavior.

In addition to the point by point slope of the parameter trends, a high-level description of the parameter behavior was used to capture the oscillatory nature or a generally upward or downward trend of the parameters which is not obvious from a point by point perspective. Therefore, two additional features were created to account for this parameter behavior when generating the random forest. A Python program was written to split each parameter into 4 or 5 point windows, depending on the total number of points in the set. The window size can be easily adjusted in the code to capture more data points in

each window if deemed beneficial. A linear regression of the points within each window was performed, calculating an associated R value falling between -1 and 1 and a slope of each resulting regression line. The R and slope values for the regression of each window were then applied to every point within the window and constituted the second and third groups of features when generating the random forest.

Along with the features, the training data contains explicitly defined labels for each point in the feature data set. The labels were defined as the UCS slope classification, where the slope of the UCS was classified on a scale of -11 to 11, similar to the parameter classification. The UCS derivative was similarly normalized to fall within the -11 and 11 scale, and then the associated class applied. With the features and labels defined, the random forest classifier can be run to generate the random forest using the test data set. The test data set, made up of features and labels, is summarized in Table 3.5. The resulting random forest prediction model then uses the features of the test data set to predict the associated labels, which can be compared to the actual test labels for accuracy.

Table 3.5: Summary of Random Forest Algorithm Features

	WOB	**ROP**	**Torque**
Point by Point	Slope Classification	Slope Classification	Slope Classification
Data Interval (Linear Regression)	R Value	R Value	R Value
	Slope of Regression	Slope of Regression	Slope of Regression

3.4.3.3 Running the Random Forest Algorithm

The random forest was generated by setting the following hyperparameters, or settings within the sklearn random forest algorithm. The first being the "n-estimators" hyperparameter, which defines the number of trees the algorithm will generate to create the forest. This was set as 1000 in the analysis presented here, but can be adjusted to change the model output. The second hyperparameter defined was the "random_state," which was defined as 42. The number defined for "random_state" is arbitrary and can be any integer. The defined integer is taken as the seed number for the random number generator in the sklearn random forest algorithm. By setting it as 42, each time the random forest algorithm is run, it will take 42 as the seed used by the random number generator and produce the same results for the given training set. If no integer is defined for the "random_number" hyperparameter, the default is to assign a randomly generated number each time, therefore the results will not be consistent with repeated implementations using the same training set. The remaining hyperparameters were left as their default settings.

3.4.3.4 Random Forest Refinement Capabilities

After generating the random forest, the sklearn software library has a function for assessing the weighted influence each feature had on improving the accuracy of the model. This is included in the program written for this work to be automatically generated each time the program is ran. Based on the results, features can be adjusted to improve how they are describing the parameter behavior to improve the prediction models. Some ways in which adjustments can be made include the smoothing window for the raw data, the thresholds for estimating constant behavior, the windows over which linear regressions are run to obtain the R and slope, and the granularity of the classification scale, such as using a -20 to +20 integer scale or -5 to +5 integer scale. If a feature does not present a significant

influence on the model accuracy, it can be removed completely, while other features can be added and adjusted. Similarly, the definition of the label can be modified to include less or more detail. For example, the granularity of the classification scale or the threshold setting for estimating equivalent constant behavior can be changed. The random forest results discussed in Chapter 4 should be considered an initial implementation of the machine learning model. The results may be used to recommend algorithm refinement through simple adjustments of the settings, or through the design of additional features and alternative labels.

Chapter 4: Results and Discussion

4.1 Manual Analysis of MSE and Parameter Trends

The MSE term was initially manually investigated as a qualitative indicator of like rock sections along the wellbore. Based on the work of Logan (2015), if DE is assumed constant, MSE changes should only be induced by a change in UCS or rock strength. Alternatively, and based on the work of Dupriest and Koederitz (2005), if rock strength is assumed constant, MSE changes should only be induced by DE changes. Therefore, it is critical to distinguish between a DE change and a rock strength change when interpreting the MSE trend. Defining a DE analytical term to model estimated DE changes and apply directly to MSE to arrive at a UCS value would be complex and discontinuous as many different influences contribute to DE and change in time. Instead, the parameter trends were investigated for identifiable behavior to distinguish, at least qualitatively, the difference between a UCS change and a DE change. The analysis yielded examples of varying parameter behaviors, under believed UCS and DE changes, which suggest a correlation may be possible between parameter trends and dominating MSE influences. However, manual analysis is highly subjective and inconsistent in describing specific parameter behaviors, leading the investigation towards automated methods for parameter description and identification.

4.1.1 MSE as an Indicator of UCS

The analysis of MSE began by calculating and plotting the MSE value along the wellbore. The MSE trend was smoothed using a 2 minute moving average, and binned, or divided into 50 ft intervals. Within each bin, an average MSE value was taken over the interval. The three forms of MSE calculated for Well 1 are plotted vs. MD in Figure 4.1.

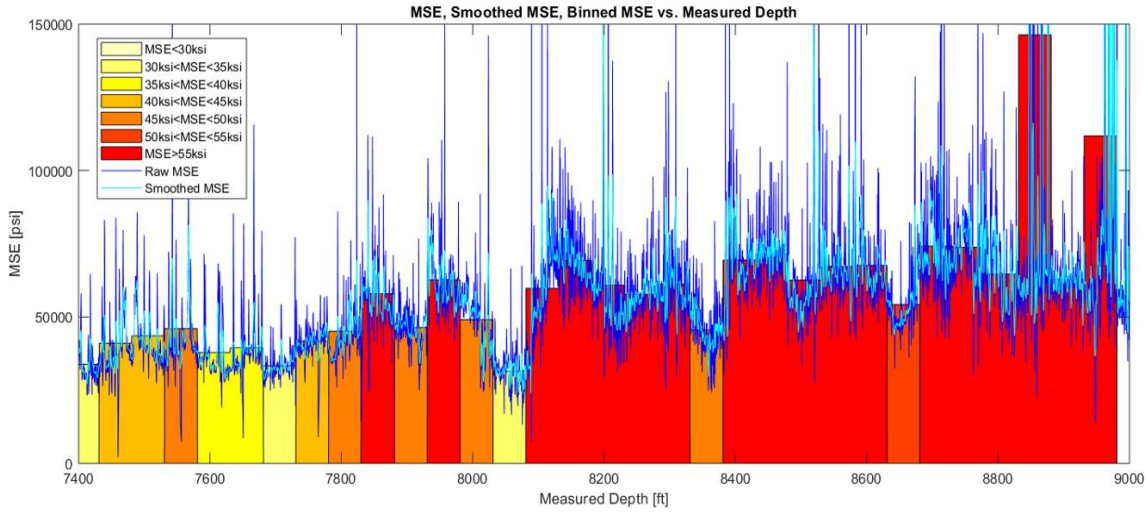


Figure 4.1: Well 1 Raw, Smoothed, and Binned MSE vs. Measured Depth

The MSE trend was then compared to the available sonic derived UCS data from an offset well. The TVD based UCS data was mapped to the Well 1 TVD reference so that both data sets could be viewed in terms of Well 1 measured depth (MD) and time drilled. Figure 4.2 shows the raw MSE calculated for Well 1 and the UCS from an offset well plotted together vs. MD. Figures 4.3 and 4.4 show the smoothed MSE data for Well 1 and the binned MSE data for Well 1 similarly plotted with the offset UCS data. As the granularity of the MSE trend is reduced from Figure 4.2 to Figure 4.4, the MSE and UCS trends tend to exhibit a better fit. The more and more generalized the MSE trend becomes, the more we see a high-level change that corresponds to the most significant area of change in the UCS trend, at approximately 8000 ft MD. This period of change may be demarked as a boundary of "like rock". This is best seen in Figure 4.4 where the lower UCS values coincide with the depths for which the binned MSE values are lowest, and highest where the UCS range has generally increased.

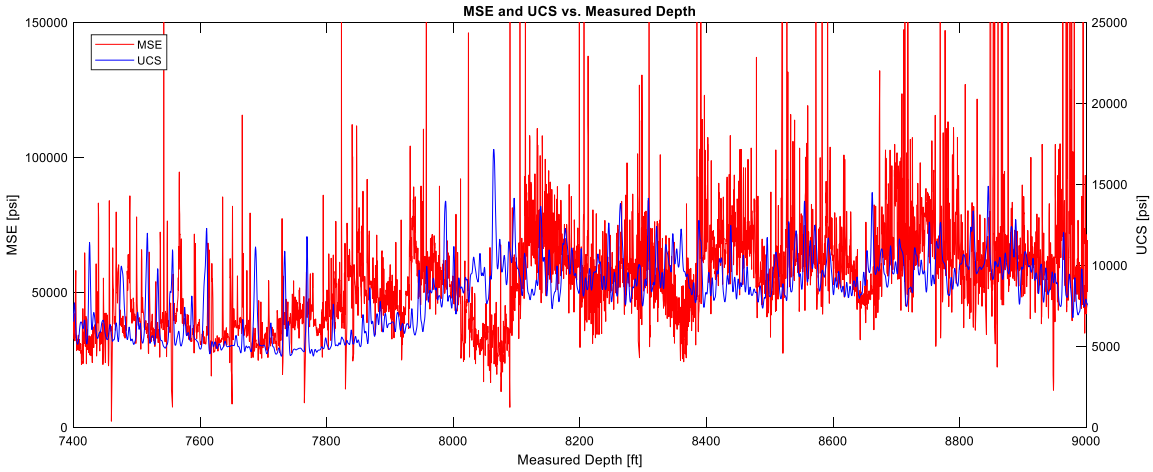


Figure 4.2: Well 1 Raw MSE and Offset UCS vs. Measured Depth

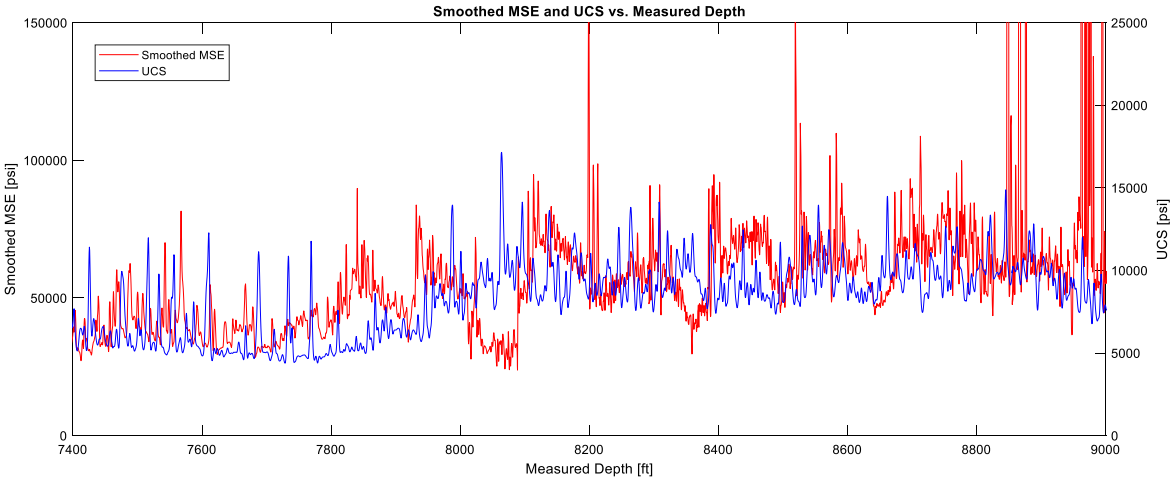


Figure 4.3: Well 1 Smoothed MSE and Offset UCS vs. Measured Depth

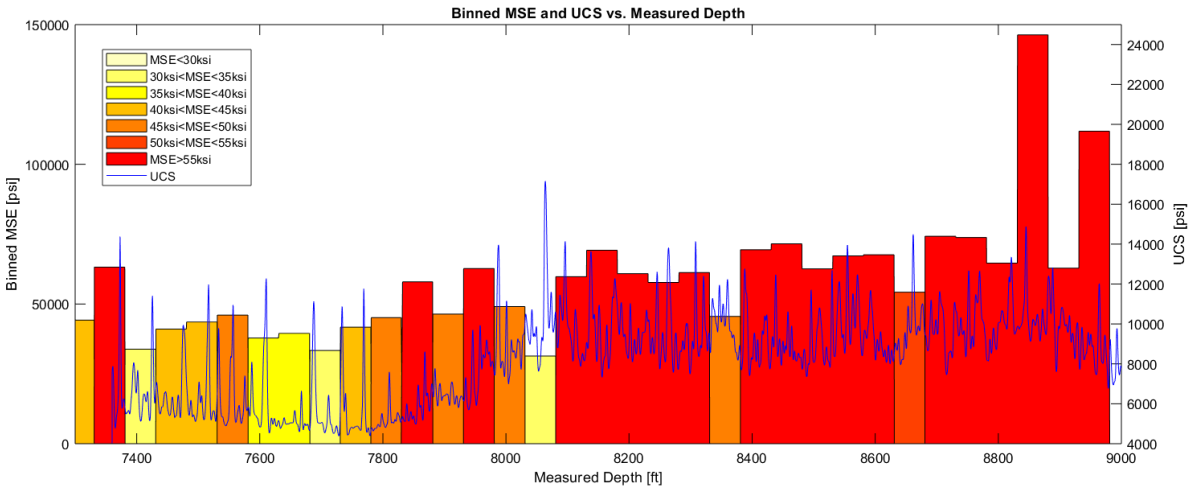


Figure 4.4: Binned MSE and Offset UCS vs. Measured Depth

A generally increasing UCS trend is typically seen with increasing depth due to compaction and other geological influences. However, the UCS trend in this case appears to transition to a higher UCS range and remain within that range over the next approximately 1000 ft. The UCS trend does not behave in a smooth way, but contains relatively frequent spikes and small-scale variability. These are potentially due to laminations in the rock which result in brief changes in the rock strength along the wellbore. The nature of the sonic measurement means the UCS data is smoothed over the length of the tool, which defines the resolution of the measurement, and therefore contains some inaccuracy in depth and magnitude of the UCS values when laminations are present. Depending on the strength of the laminations, the small scale UCS variability would be expected to produce an MSE response if the bit is drilling efficiently.

The calculated MSE term in Figures 4.2, 4.3, and 4.4 exhibits an overall trend that increases with depth, relatively matching the UCS trend at a high level. In addition to the

influence of UCS changes, this trend may be in response to increasing wellbore friction with depth, progressive bit wear, or other DE related influences that tend to accumulate with depth. Although the high-level match between the MSE and UCS trends is somewhat discernable in the raw data in this case, there appear to be some inaccuracies on a scale of interest. As the comparison becomes more granular, the inaccuracies increase. While the depth references may contain some inaccuracy due to lateral variability between the offset UCS data and the well drilled, direct comparison of the UCS and MSE trends at a small scale tends to exhibit little correlation. A small scale can be as large as 50 ft, as is evident in Figure 4.2. Overall, the initial evaluation of MSE as an indicator of like rock suggests that the depth scale of the analysis and degree of smoothing or generalization are significant factors in the accuracy of the interpretation.

4.1.2 Parameter Trend Analysis

To understand small-scale response to UCS changes in the MSE term, it must be known whether a UCS change or a DE change is causing the MSE change. In an attempt to distinguish when one of these two influences is dominating the MSE response, the analysis shifted to the parameters which define the MSE term. Figure 4.5 shows the Well 1 drilling parameters (ROP, WOB, and torque), MSE, inclination in degrees from vertical, and offset UCS data plotted vs. cumulative time in hours in a zoomed in section of the wellbore, approximately 270 ft MD. At approximately 84.7 hrs, a break in the data occurs where a drill pipe connection is made. It is important to recognize where drilling breaks occur as the ramp up period immediately following a drilling break is a dynamic efficiency period where MSE should not be considered reliable. A drilling period could be between connections of drill pipe or a pause in the active drilling operation due to tool face

adjustment, back reaming, or other reason for a pause in continuous WOB and RPM application. These are identified by the breaks in the data in time, two of which are evident in Figure 4.5. RPM was not included in the example presented in Figure 4.5 as the section was drilled using rotary drilling with a relatively constant RPM. Where brief RPM adjustments occurred, no direct influence was observed between the new RPM value and other drilling parameters.

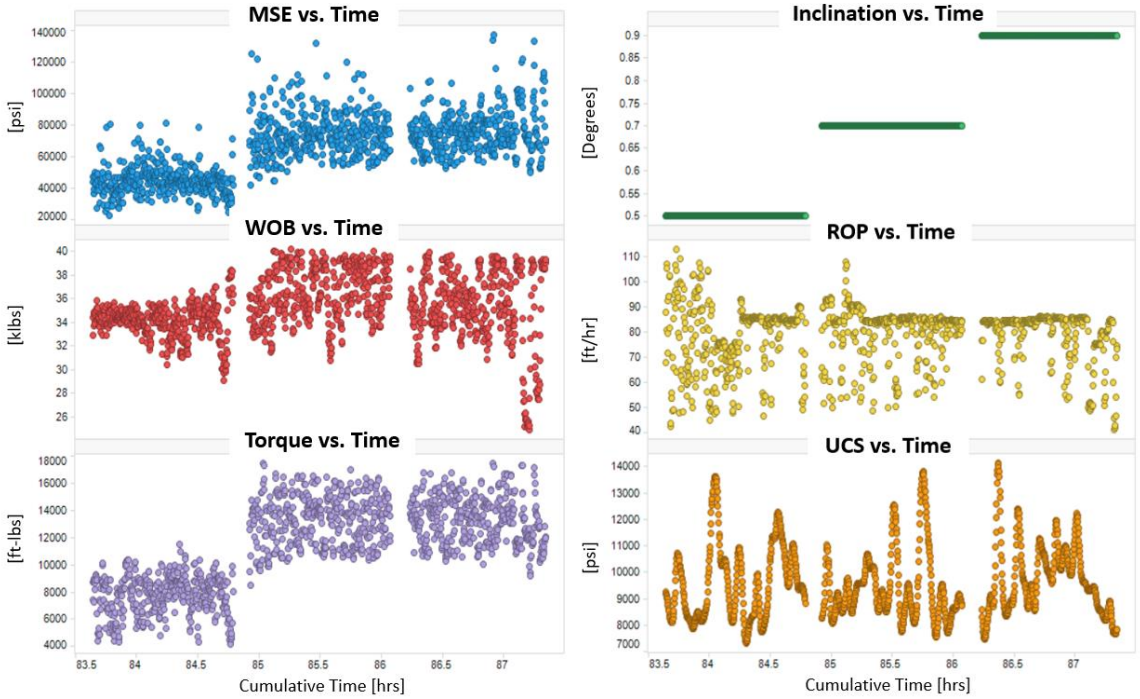


Figure 4.5: Well 1 MSE, Inclination, WOB, ROP, Torque, and Offset UCS vs. Cumulative Time

The duration of both drilling periods is the time to drill a single stand of drill pipe into the hole, in this case 90 ft MD. When drilling recommences after the connection at approximately 84.7 hrs, the MSE value begins at a higher value than the previous drilling period. Looking at the inclination plot, the tool face was approximately vertical over the

entire section. Looking at the parameter contributions to MSE, WOB was increased by approximately 5,000 lbs which resulted in a significant torque increase. The torque change was not accompanied by an ROP increase, therefore a WOB change induced an MSE change. This implies that the bit is not efficient, per the work of Dupriest and Koederitz (2005) presented in Chapter 2, and that the increase in WOB decreased the overall average DE. Due to the relatively stable nature of the MSE at this scale, this MSE trend may incorrectly be generalized to described a 180 ft section of "like rock" without taking into account this change in DE.

To evaluate the relative parameter relationships in more detail, the parameters were cross plotted, as shown in Figure 4.6. Figure 4.6 shows the data points associated with only the third drilling period from the data shown in Figure 4.5. The data points highlighted in Figure 4.6 are associated with the highest range of MSE values. The torque and ROP values increase relatively linearly with increasing WOB up to approximately 31,000 lbf WOB. Beyond this WOB, the ROP reaches a relative maximum and begins to drop significantly at the highest WOB values reached. Similarly, beyond 31,000 lbf WOB, the torque begins to fluctuate within a wide range, and exhibits low values at the highest values of WOB. The highlighted data points show that the MSE peaks occur primarily when WOB is above this threshold. This behavior indicates that beyond 31,000 lbf WOB, a founder is reached and the DE changes. A large portion of this drilling period occurs at the non-optimized WOB level, creating MSE spikes that do not correspond to the small scale UCS trend. The parameter behaviors beyond the WOB threshold of 31,000 lbf may fall within Region III of the drill off curve, as described in Chapter 2, and 31,000 lbf may be defined as the WOB founder point in this drilling state. As shown in Figure 4.5, WOB fluctuates above and

below this founder point repeatedly over the third drilling period, therefore the magnitude of inefficiency is constantly changing.

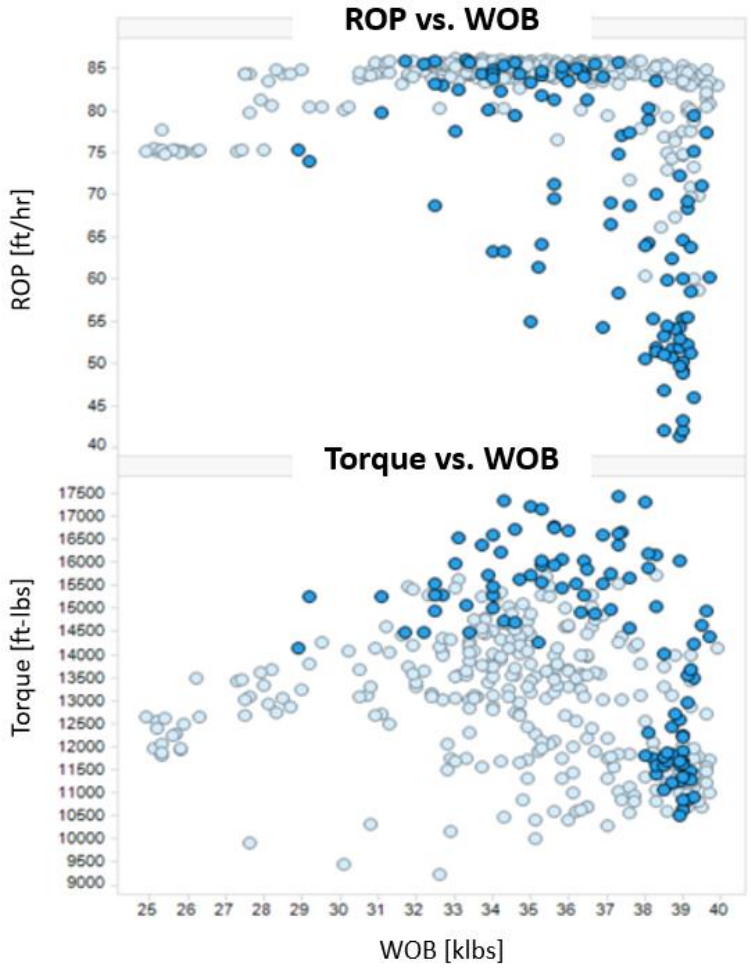


Figure 4.6: Well 1 ROP and Torque vs. WOB

The data from the first period in Figure 4.5 is plotted with the third period, and presented in Figure 4.7 with the data from the first period highlighted for reference. Two distinct data clouds emerge within the cross plots in Figure 4.7 due to two different average, or baseline, efficiencies occurring between the two stands. This is also evidenced by the

average MSE values observed in these two nearby periods. The magnitude of the prevailing ROP and Torque ranges between two nearby stands has changed. Therefore, in this example, a step change is observed in the generalized DE from one stand to the next that occurs concurrently with smaller scale continuous fluctuations in DE due to the WOB founder.

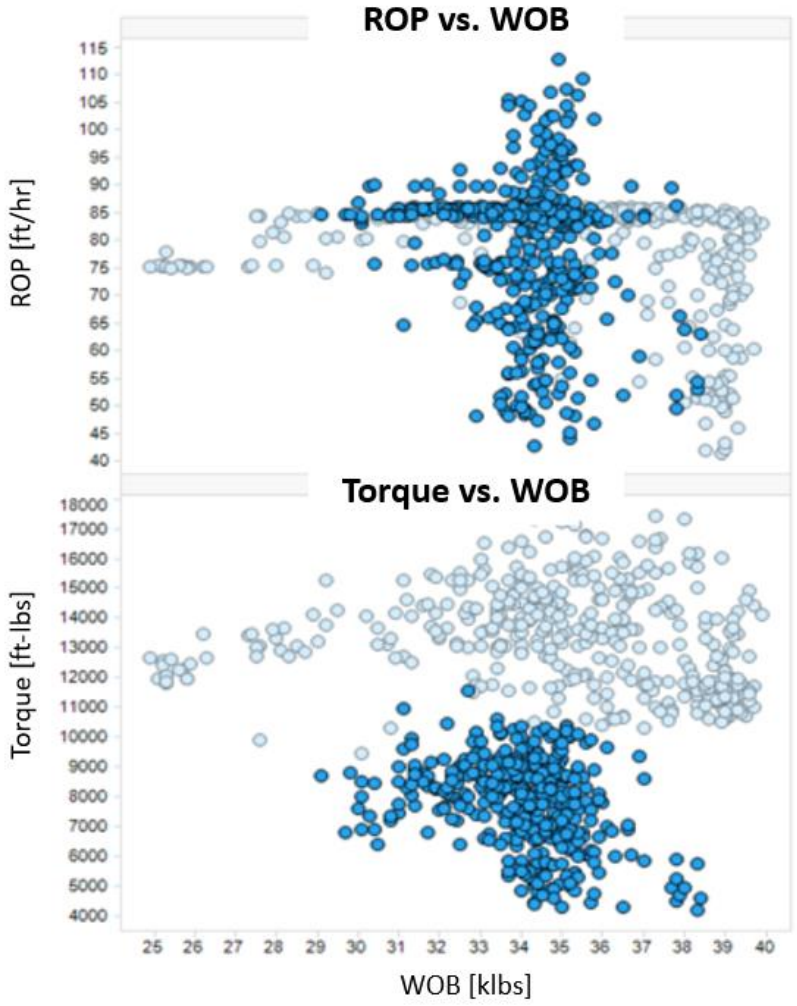


Figure 4.7 Well 1 ROP and Torque vs. WOB, Two Drilling Periods

An investigation of the drilling parameters reveals that a DE influence may be dominating the MSE trend in most areas. This suggests caution is necessary when interpreting MSE for areas of "like rock" under the assumption that DE is constant. At a high level, it is observed in this case that an increase in WOB resulted in an increase in torque without a corresponding increase in ROP, which may represent a combination of parameter trends to indicate DE as the dominant influence. However, this response becomes less distinguishable as the scale is reduced.

4.1.3 Well to Well Comparison

In an attempt to identify likely areas where rock strength change is dominating the MSE trend, the MSE and parameter data was compared between the wells to look for similarities in the trends observed at surface. If the MSE trend observed at surface in one well is indicative of UCS changes along the wellbore, then similar trends should be observed within each well on the pad as well, with some potential lateral variability. To visually represent the 3-D lateral offset between two wells on the pad analyzed, Figures 4.8 and 4.9 show the wellbore trajectories for Well 0 and Well 1. Figure 4.8 is a heel side view, showing the sideways offset between the two vertical sections and landing points of the laterals. Figure 4.9 shows a rotated view, with a clear image of the forward offset and directional nature within the vertical section of Well 0 compared to Well 1.

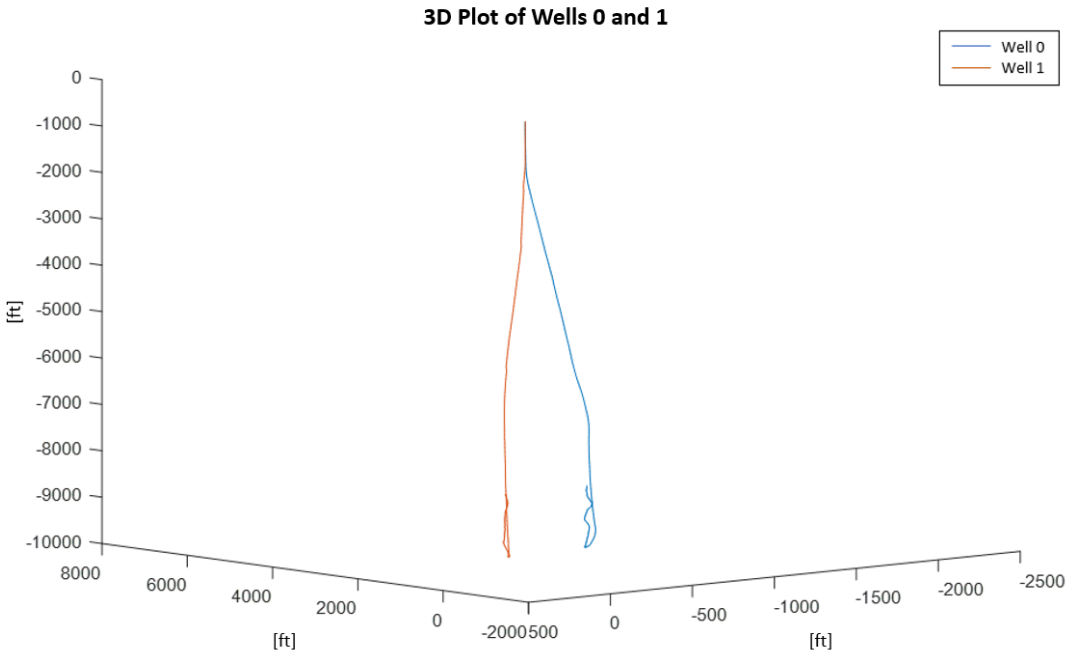


Figure 4.8: Well 0 and Well 1 3D Well Trajectories, Heel-Side View

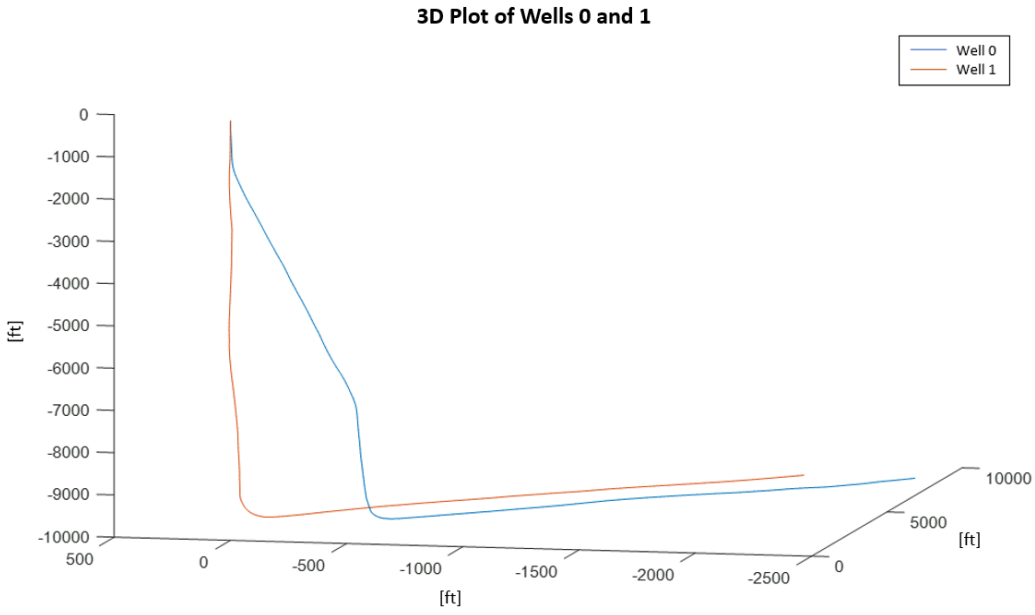


Figure 4.9: Well 0 and Well 1 3D Well Trajectories, Rotated Side View

The MSE values for Well 0 and Well 1 were plotted vs. TVD, as well as the offset UCS data used in the analysis discussed in the previous section. The offset UCS data is assumed to contain some TVD variability when compared to the MSE data due to lateral lithology changes between the offset well location and the pad location. Therefore, a direct comparison to the MSE data is not necessarily accurate. Comparing MSE to MSE between two wells was a potential method for secondary validation of the MSE trend for indications of UCS changes. The offset UCS data used in this analysis appeared to best fit the Well 1 MSE data, and therefore Well 1 was considered a good candidate for baseline comparison with other wells. However, the analysis could be carried out with no UCS data as similarities between MSE trends are unlikely coincidental and potentially indicate UCS influence in both wells. Figure 4.10 shows the Well 0 and Well 1 MSE data, shown in red and blue respectively, plotted with the offset UCS data in green. Figure 4.11 shows the ROP, WOB, and torque for the same section of each well shown in Figure 4.10. ROP is shown in orange, WOB in purple, and torque in blue. A wellbore section of approximately 250 ft is shown.

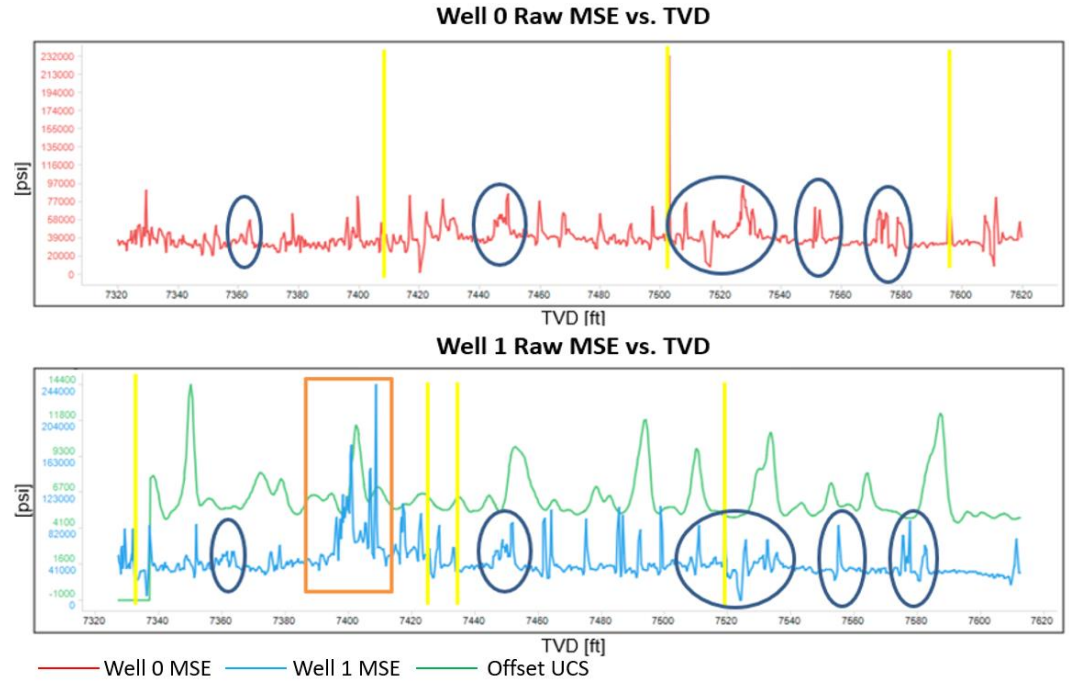


Figure 4.10: Well 0 MSE, Well 1 MSE, and Offset UCS vs. True Vertical Depth

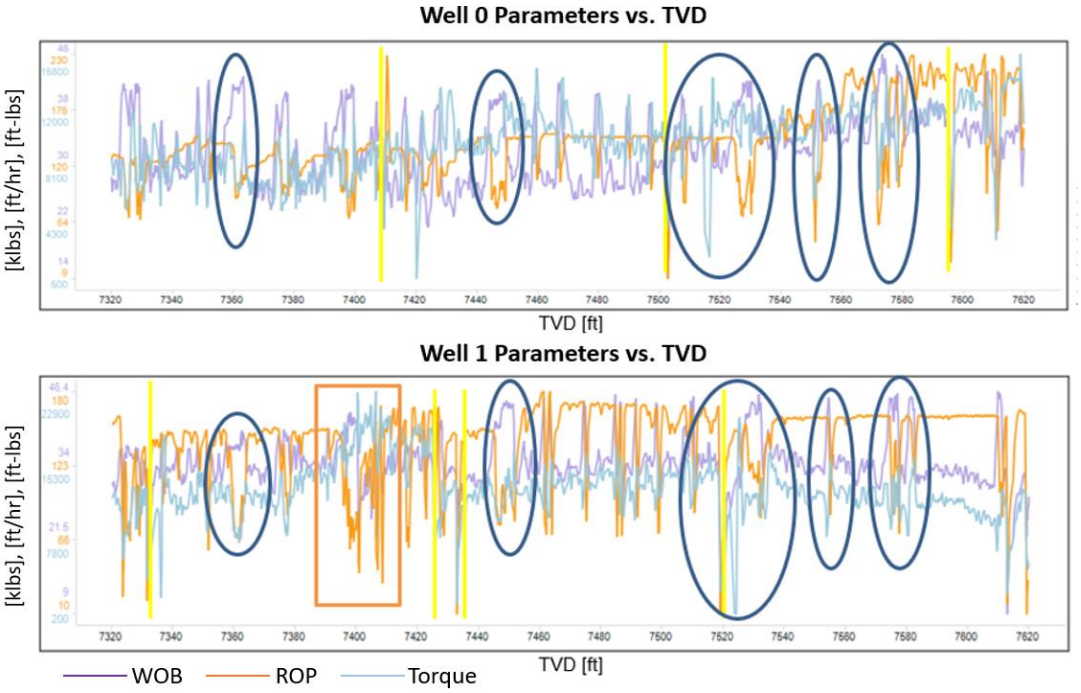


Figure 4.11: Well 0 and Well 1 WOB, ROP, and Torque vs. True Vertical Depth

The well to well comparison between Well 0 and Well 1 shows instances of MSE behavior at similar TVD that appear to match between the two wells. These are indicated by circles in Figure 4.10. The MSE is plotted in its raw form, after data cleaning, and was not subjected to any smoothing. At approximately 7450 ft TVD, we see a similar shape in the MSE trends in both wells, appearing to increase, briefly maintain the increase, then locally spike before returning to the previous baseline. We see similar shared patterns at about 7360 ft TVD, 7520 ft TVD, 7550 ft TVD, and 7580 ft TVD. The offset UCS data indicates local UCS increases periodically along the wellbore, some of which may correspond to the observed common MSE patterns. However, due to the lateral variability potential between the pad and the offset UCS data, some TVD error is expected in the UCS data. The well to well MSE comparison provides a more compelling case for lithological influence on the two MSE trends.

It is important to note where the drilling breaks occur in the well to well comparison. Figure 4.10 indicates the locations of these breaks with vertical yellow lines. Both wells exhibit instances where the initial points in a drilling period experience high MSE values. This is expected based on the previously discussed ramp up period at the beginning of each drilling period. However, it is not observed at the beginning of every drilling period. The instances of high MSE at the beginning of the drilling period should not be considered when comparing the MSE trends between wells.

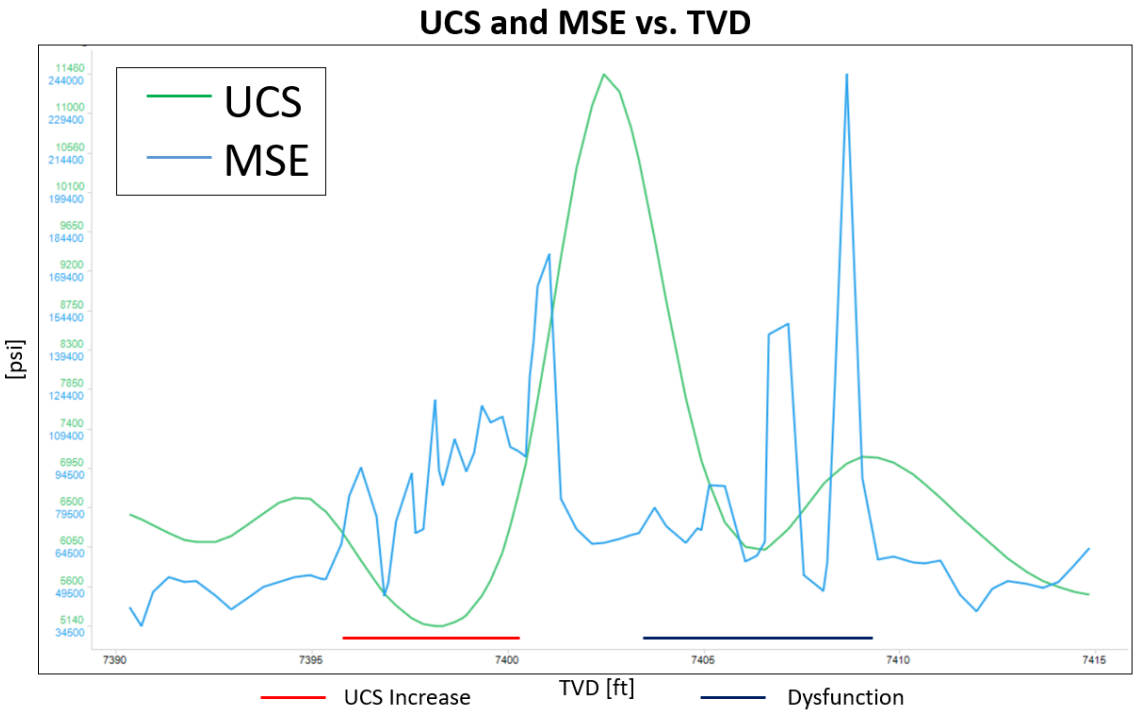
Looking at the parameter behavior between the two wells in Figure 4.11 at these TVDs, we see similar behaviors. The locations where MSE trends matched between the two wells are again identified with circles and the drilling breaks identified by vertical yellow lines. Similar changes in parameter behavior appear to be taking place at the same TVDs within both wells. Both positive and negative correlations are seen between WOB

and ROP at the locations of MSE change shared between the two wells. As previously discussed, the most likely shared influence at these TVDs is the rock strength and its lateral continuity between the two wells. Negative correlations between WOB and ROP, typically associated with DE changes, are observed with UCS changes at this scale potentially due to a transition period where the parameters adjust to the new rock strength. The positive correlations observed are more often instantaneous and may be induced by laminations, for example between 7450 and 7500 ft TVD. Again, we see some indication that parameter behavior analysis may be able to distinguish between a DE or UCS influence, however the scale of the analysis is again significant. Additionally, in this example, the classical definition of a negative correlation as a DE influence breaks down.

4.1.4 Parameter Analysis: Dysfunction Example

The well to well analysis presented in the previous section appeared to contain an instance of dysfunction at approximately 7407 ft TVD in Well 1, as shown in Figure 4.10 and 4.11, indicated by an orange box. This is a location where similar MSE behavior occurred in both wells, while the MSE change was more significant in Well 1 than Well 0. Figure 4.12 presents this dysfunction event in detail, showing the MSE and UCS plotted together and the corresponding drilling parameters vs. TVD. As shown in Figure 4.12, this location immediately follows a rise in MSE, followed by a sharp fall and period of oscillations. Additionally, the offset UCS data appears to show an instance of high UCS relative to the UCS measured around that location that occurs at approximately the same TVD as the initial rise in MSE. The MSE and UCS trends suggests this is a moment of stick slip dysfunction in Well 1. The characteristic rise and oscillatory pattern in the MSE in this case is consistent with stick slip vibration in response to the WOB as the bit re-

enters the lower UCS rock. Initially, the energy at the bit adjusts to the harder rock, as indicated by the increasing MSE and negative correlation between ROP and WOB. When the bit re-enters the lower UCS rock, the thrust energy at the bit is too large, resulting in too large of a bite and triggering the onset of stick slip vibration. As the bit energy adjusts back to the lower UCS rock, the stick slip vibration is remediated.



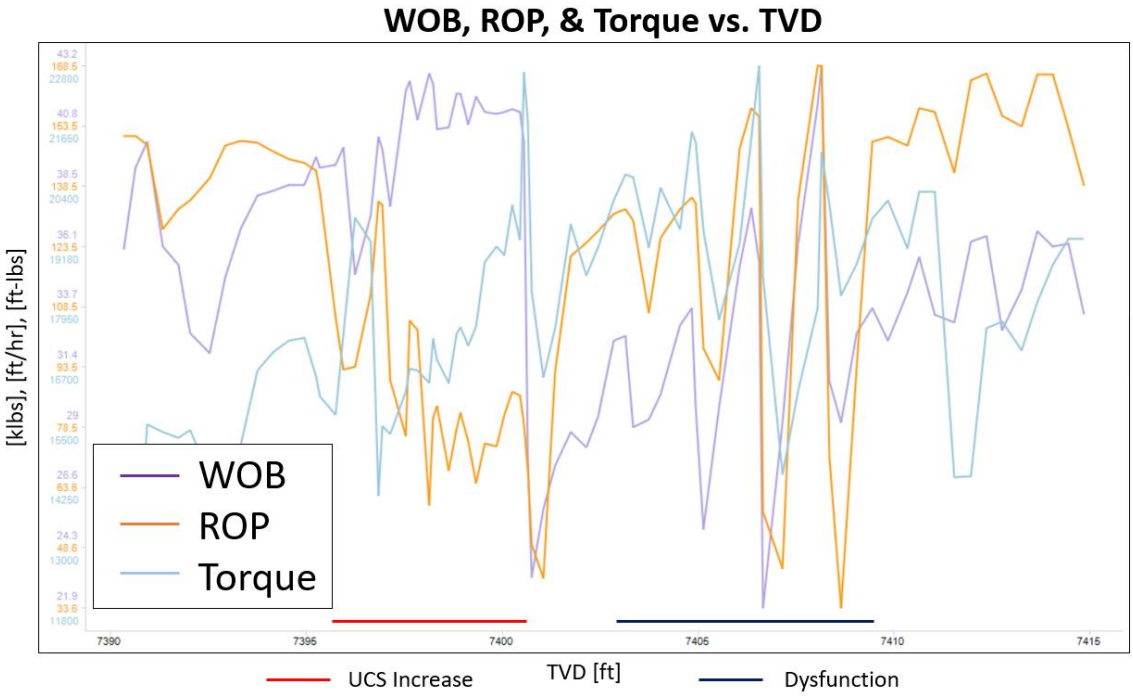


Figure 4.12: Well 1 Stick Slip Dysfunction

In this example of dysfunction, we see an MSE change due to an initial UCS increase which is immediately followed by a UCS decrease. A dysfunction induced DE change occurs either simultaneously with or immediately following the UCS decrease. Looking specifically at the parameter trends at this location, presented in Figure 4.12, we see WOB and ROP exhibit a negative correlation and torque generally increase in the area where UCS is believed to be increasing, indicated by a red bar along the x-axis. This is followed by a positive correlation between all three parameters in the oscillation period where dysfunction is believed to be taking place, indicated by a blue bar along the x-axis. Because a positive correlation is generally expected when DE is constant, a positive correlation while stick slip may be occurring is an unexpected result. The nature of the oscillations may be a defining characteristic in the parameter trends for identifying DE

changes due to stick slip dysfunction, in addition to the relative correlation of the parameters. In addition, identification of specific dysfunction types through parameter behavior may be better defined in terms of time rather than depth. Looking only at the positive correlation between the parameters at this scale, the MSE increase during dysfunction in this case may be mistaken for constant DE and thus inaccurately suggest a change in UCS.

Conversely, the increase in UCS appears to induce a parameter response indicative of inefficient drilling. However, DE did not necessarily change with the increase in UCS. The UCS increase changed the required energy at the bit to fail the stronger rock. The WOB and ROP negative correlation is observed as the bit energy changes. Intuitively, the bit will drill slower through harder rock and require more WOB. WOB and ROP appear to briefly stabilize at new baseline values before entering the lower UCS rock again. Torque however continued to adjust upwards to accommodate the new rock strength, and may not have reached a baseline before the rock strength changed again. Therefore, the correlation observed may be a dynamic energy state at the bit as the energy state changes with rock strength, exhibiting a delayed and/or transition response at the surface measured parameters. The positive MSE response in this case may be considered a reliable indication of a change in UCS.

4.1.5 Conclusions from Manual Data Analysis

The manual analysis presented in this section indicates small scale and high level parameter behaviors which suggest a correlation between unique parameter responses and DE or rock strength changes is possible. The linear parameter relationships indicative of efficient drilling did not necessarily hold true when UCS was changing on a small scale.

Logan (2015) assumes a constant baseline DE, however this may not be realistic unless the MSE trend is highly generalized. When an MSE change is triggered, distinguishing which influence is dominating, either a DE change or UCS change, is necessary to avoid incorrect interpretation of the MSE trends. Changes in DE may falsely indicate changes in UCS when interpreting the MSE trend alone. Additionally, one may expect UCS and DE to change simultaneously. Analysis of the individual parameters in combination with the MSE enhances the interpretation process, however the manual process remains subjective and inconsistent. To enhance the consistency with which parameter behaviors are defined and interpreted, automated parameter behavior definition and identification methods were implemented and evaluated, as described in the following section.

4.2 Automated MSE and Parameter Behavior Analysis

Based on the manual interpretations of MSE and parameter trends presented in the previous section, automated methods for classifying the dominant influence on MSE, either DE or UCS changes, were investigated to reduce the subjective and inconsistent nature of the manual interpretation. Where a DE change is dominating the parameter response, interpretation of the UCS trend from MSE would be unreliable. Where DE changes are not dominating and the MSE term is deemed a reliable UCS indicator, the UCS influence would be interpreted qualitatively as either increasing, decreasing, or constant.

The analysis focused on the parameter trends which define the MSE term, namely WOB, ROP, and torque, and attempted to identify unique behaviors that could be flagged as indicators for DE changes vs. UCS changes. The RPM trend was ignored as it remains fairly constant, with changes occurring relatively instantaneously. The Well 8 drilling data and Well 8 UCS data were used in this analysis to minimize inaccuracy in the UCS to MSE

comparison. The Well 8 UCS data was obtained in a vertical pilot hole section of the Well 8 well, and was then mapped to the MD reference within the Well 8 drilling data, as described in Chapter 3.

An initial set of hypotheses describing parameter behaviors were defined and analyzed. These hypotheses were based on the positive correlations typical of efficient drilling, as described by Dupriest and Koederitz (2005), and are summarized in Table 3.3. These correlations assume UCS is constant, therefore the locations where they occur were reviewed as potential indicators of constant UCS. To test these hypotheses, locations where the derivatives of WOB, ROP, and torque matched (all positive, all negative, or all constant) were identified using the programmatic methods described in Chapter 3. Figure 4.13 shows the results where the MD locations of these slope combinations are marked in red. The results are based on a point by point derivative of the raw parameter data. All other data points, shown in blue, are locations where the slope combinations described did not occur. The top plot is the MSE along the Well 8 wellbore and the bottom plot is the mapped UCS values obtained on Well 8, both of which are filtered to show the locations of the parameter behavior as described.

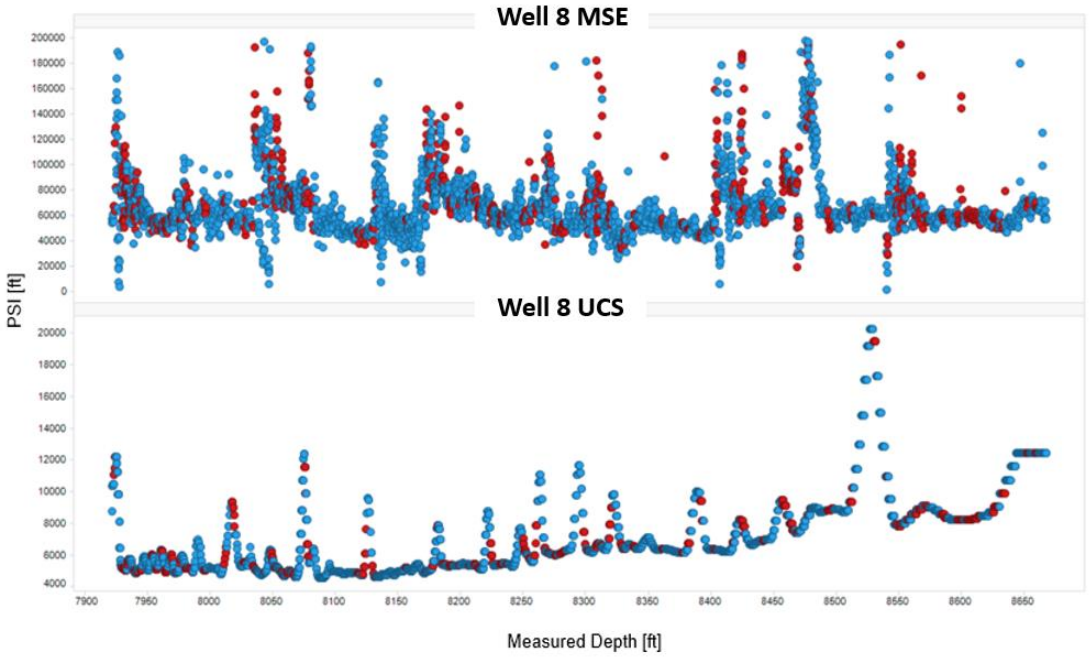


Figure 4.13: Locations of Hypotheses for Parameter Response with Constant UCS

The results show a weak correlation between constant UCS and the parameter slope combinations as defined by these hypotheses. At approximately 7950 ft MD, the UCS data exhibits oscillations within a range of 2000 psi. When looking at the results shown in Figure 4.13, the peaks and troughs in those oscillations appear to exhibit constant UCS behavior. At each peak and trough, a relative baseline is potentially reached where the bit has adjusted to the new UCS and stabilized, seen at a small scale. Similarly, in some cases the parameter combinations are observed at or near the high peaks in the UCS data. At deeper MD locations, we see data classified as constant UCS more often in areas where UCS is relatively stable than where peaks or significant transitions occur. However, the

duration and frequency of these slope combinations are limited, leaving a significant amount of data unclassified where they do not occur.

In an attempt to describe more of the data along the wellbore, the slope combination hypotheses were expanded beyond the classical efficient behavior described by Dupriest and Koederitz (2005), where UCS is assumed constant. A set of additional parameter slope combinations were defined based on an idealized parameter response as the bit moves from one rock strength to another, or while UCS is changing. Furthermore, the driller controlled WOB input may or may not be changing at the same time. RPM is again ignored and assumed constant. ROP and torque may be thought of as "responses" to the "inputs" of WOB and UCS. Intuitively, as the bit begins to cut harder rock, if WOB is held constant, ROP should exhibit a drop accompanied by an increase in torque. Similar behavior was observed in the manual data analysis previously discussed, specifically in the dysfunction example where UCS initially increased. Conversely, if UCS is decreasing with constant WOB, the response should be the opposite. Complexity is introduced when WOB and UCS are changing together. Therefore, the parameter combinations included in the new set of hypotheses were simplified to the case where an increase in UCS with a simultaneous increase in WOB will completely offset the ROP drop due to the higher rock strength, resulting in a constant ROP response. Additionally, in the case where UCS and WOB are moving opposite one another, the torque change is negated. The full set of hypotheses tested in this stage of the analysis are presented in Table 3.4.

Figures 4.14 and 4.15 show the Well 8 MSE and UCS data filtered to indicate the MD locations where the parameter slope combinations presented in Table 3.4 occurred. Figure 4.14 shows the results of the analysis using the point by point derivatives of the parameters. Figure 4.15 shows the results using a binned definition of the parameter

derivatives, where the parameters were divided into 1 minute bins and a slope estimated over the bin. The binned slope was applied to every point within the bin and then used to identify slope combinations, as described in Chapter 3. Red indicates where UCS is increasing, blue where UCS is decreasing, and green where UCS is constant, per the UCS slope hypothesis associated with each parameter response as defined in Table 3.4. Yellow represents all data points where none of the defined slope combinations occurred. With three parameters and three possible slopes for each, there are 27 possible slope combinations to define the data set, only 9 of which were included in the new set of hypotheses.

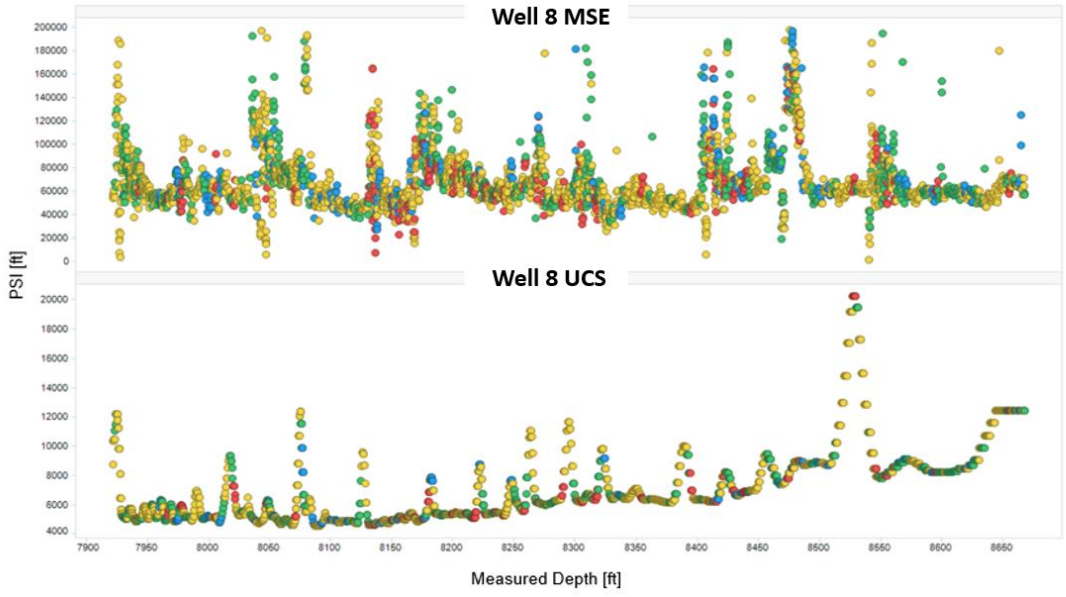


Figure 4.14: Locations of Expanded Hypotheses for Parameter Response to UCS, Raw Derivative Analysis

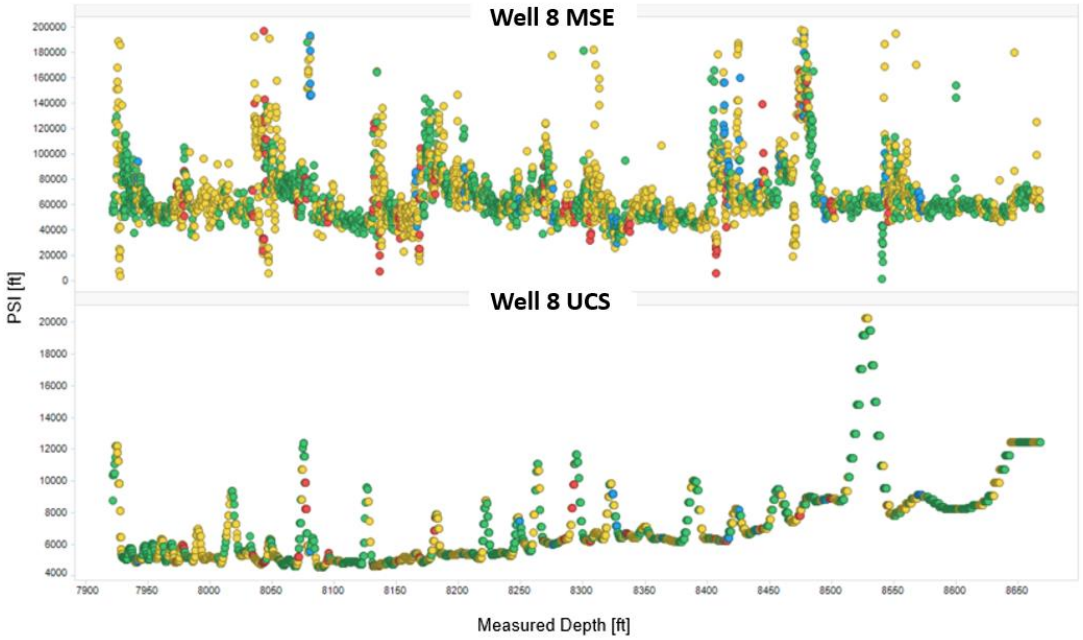


Figure 4.15: Locations of Expanded Hypotheses for Parameter Response to UCS, Binned Derivative Analysis

The results show a significant amount of data is again not classified by the expanded set of hypotheses, indicated in yellow. The results using point by point derivatives, shown in Figure 4.14, appear to accurately capture UCS change in some areas of constant UCS, as well as areas where UCS is increasing or decreasing around relative peaks. For example, at approximately 8075 ft MD, we see a large portion of downward slope of the relative peak in UCS at this location correctly classified as decreasing UCS. Additionally, both sides of the relative peak at approximately 8175 ft MD appear to contain some instances of accurate UCS slope classification. However, the majority of the classifications are inaccurate in both areas of primarily constant UCS and changing UCS. The thresholds set to define equivalent constant behavior may account for some instances where constant UCS is inaccurately indicated.

The results from the binned parameter slopes, shown in Figure 4.15, appear to be less accurate, with more data classified as constant UCS. This is reasonable as the baseline UCS trend increases gradually over the section shown, increasing in total by approximately 4000 psi. The generalized slope is thus indicating a relatively constant UCS over much of the data. The results may provide a similar interpretation of UCS to that obtained directly from the MSE term in its most generalized form, where the binned MSE trend matched the high level UCS trend.

The hypotheses used in the analysis do not account for a change in DE. Therefore, where the classifications are inaccurate, a DE change may be taking place. When UCS is changing, a UCS and DE change may be happening simultaneously. If DE is dominating the parameter behavior in these instances, additional descriptive detail may be required to characterize the parameter response as a DE change vs. UCS change. The manual process for assigning a hypothesized UCS influence becomes more labor intensive the more detailed the description of the parameter behavior becomes. Attempting to manually assign a DE change to a parameter behavior increases the complexity as well. Therefore, alternative methods for assigning and testing a hypothesized UCS influence to a parameter behavior were explored.

4.2.1 CBM Implementation Results

The hypotheses analyzed in the previous section were based on a manually assigned UCS influence. The results showed significant amounts of unclassified data representing the remaining parameter slope combinations that were not assigned hypotheses on what UCS changes, if any, were influencing those parameter behaviors. To expedite the hypotheses, or prediction assignment process, a statistical algorithm was written to identify

the most frequent UCS slope associated with each parameter behavior. This algorithm is based on the assumption that for the majority of the data used in this analysis, DE was relatively constant, or non-dominant in its influence on parameter behavior. The algorithm builds a CBM, as described in Chapter 3, that defines the parameter slope combination, as well as the combination of magnitudes of the parameter slopes at each point, or over a bin. The CBM is then used to identify the associated UCS slope at every instance of a unique behavior within the CBM, and assigns the most frequently occurring UCS slope to that behavior. If no UCS slope occurs more than 50% of the time when that behavior occurs, the behavior is left unclassified. The prediction assignment obtained from this algorithm was then applied back to the data set, identifying the locations where a positive, negative, or constant UCS prediction occurred, and where no classification was given. The results of the CBM predictions are shown in Figure 4.16, plotted vs. MD. The data points shown in red indicate locations where a prediction of increasing UCS slope was assigned, blue where a decreasing UCS slope was assigned, and yellow where no prediction was assigned.

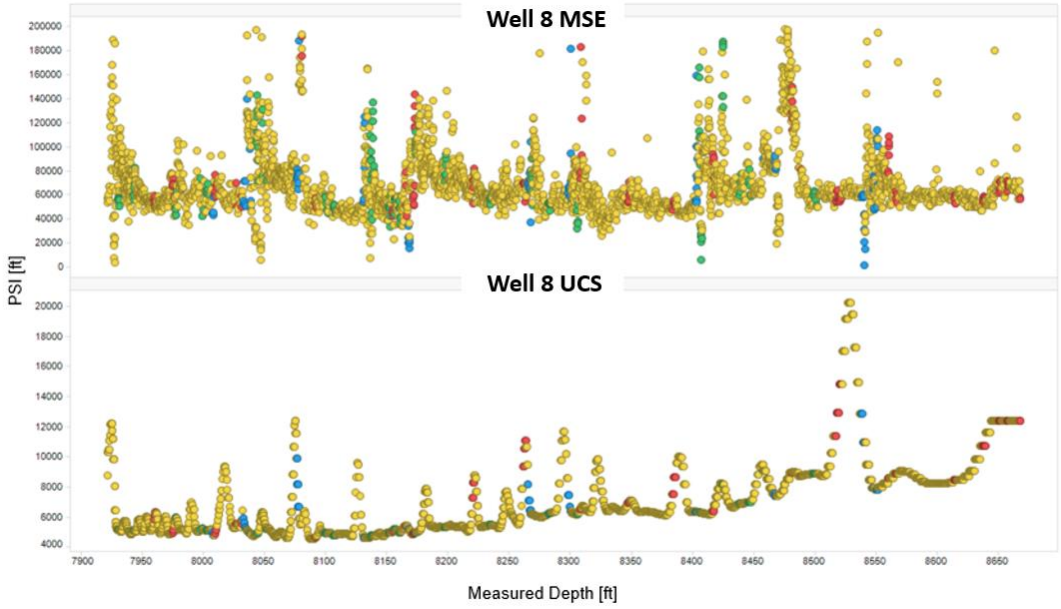


Figure 4.16: Locations of CBM Predictions

Overall, the CBM classified only a small amount of data. The parameter behavior as classified by the CBM may be associated with more than just a UCS change, i.e. a DE change or combination of UCS and DE change. Where the CBM prediction did find a classification, the prediction appears to be fairly accurate. This suggests that the current CBM description of parameter behavior is sufficient in these instances to describe a dominant UCS influence. Additionally, these instances occur when DE changes are not dominant. If these behaviors occurred elsewhere in this data set where DE was dominant, the classification would likely be inaccurate. Because we do not see this, the classified parameter behaviors may be specific to UCS changes.

There appear to be some instances where the UCS is classified as increasing when the UCS is not significantly changing. Inaccuracy in classifying UCS as constant may be

due to the threshold settings which define equivalent constant behavior. The labels of increasing UCS could be considered either constant or increasing depending on the granularity of the definition.

If the CBM prediction results in Figure 4.16 can be considered instances of dominant UCS influence on the parameter response, then the associated MSE values may also be considered reliable locations for indication of the UCS trend. However, the locations where these occur in the MSE trend do not necessarily track with the UCS trend. Figure 4.17 highlights the CBM classified data within the MSE and UCS trends for emphasis. The MSE trend where classification occurs does not appear to be a good indicator overall of the UCS trend across these locations. Therefore, some underlying DE influence may be taking place to inflate the MSE term between these instances, shifting the MSE trend up or down in a discontinuous way. A detailed estimate of the DE influence on the MSE term at all times would be required to convert it into a trend that tracks with the UCS. Instead of attempting to directly correlate the MSE trend to the UCS trend, the characteristic parameter behavior may be a better qualitative indicator of UCS movement than the MSE trend itself.

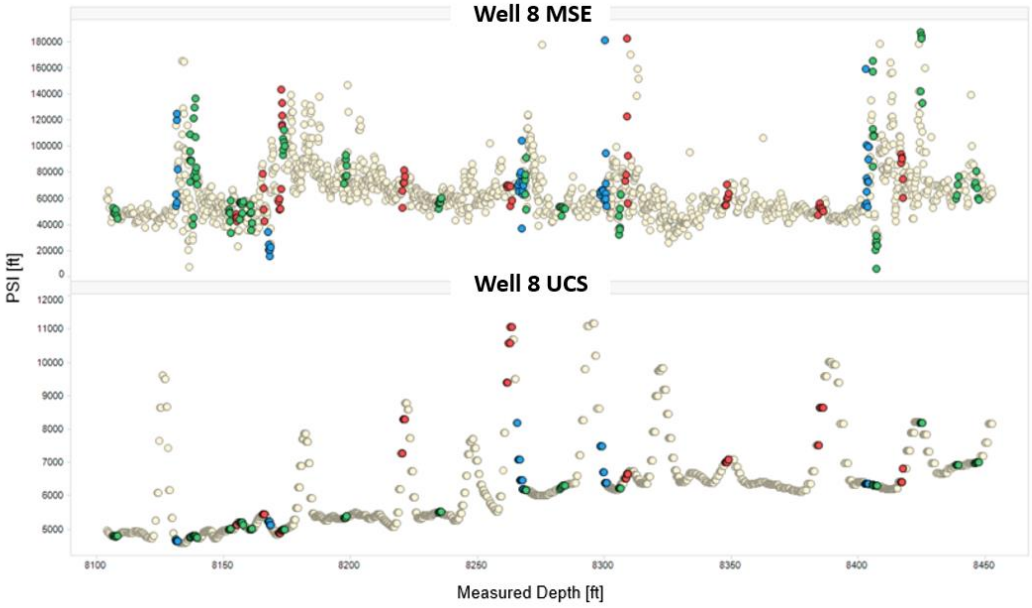


Figure 4.17: Locations of CBM Predictions Highlighted

4.2.2 Random Forest Implementation Results

To enhance the process of assigning UCS slope predictions to specific parameter responses, a machine learning method, Random Forest, was implemented. The Random Forest algorithm is given features, or descriptive details of the parameter, and uses them to create Decision Trees based on subsets of the data. The aggregated set of decision trees informs a prediction, the Decision Trees themselves making up the Random Forest. The implementation methods are described in detail in Chapter 3.

The first set of features are the point by point derivatives of each parameter. The derivative is classified at each point based on its magnitude, using a scale from -11 to +11, thereby capturing the magnitude of the change as well as its direction. The second and third sets of features are based on a window of data where a linear regression is performed using the points within the window. The linear regression produces an R value between -1 and 1

indicating the degree of oscillations or scatter within the window. Using the R value instead of the R^2 value allows the feature to maintain the regression directionality in the assessment of scatter. The slope of the linear regression line is the final feature set. This provides a description of the high level trend within the data, as opposed to the instantaneous point by point derivative. Using these three sets of features, the Random Forest model predicts a UCS response at each point, defined on a scale between -11 and +11 similar to the point by point derivate feature.

The Random Forest model is generated using a subset of data from the Well 8 data set, including the Well 8 UCS data, and tested on the remaining data. The data was split into train and test sets in two ways: splitting the data in half based on MD, and randomly selecting 75% of the data as the train set and 25% as the test set. The results from both analyses are presented in Figures 4.18 and 4.19. The plots in Figure 4.18 show the predicted UCS slope for the test set using the Random Forest models in red, and the actual UCS slope of the test set in blue. To further visualize the prediction accuracy from both models, the predicted UCS slope and the actual UCS slope of the test data sets in each case were cross plotted, as shown in Figure 4.19. The linearity between the predicted and actual UCS slopes indicates a higher degree of accuracy.

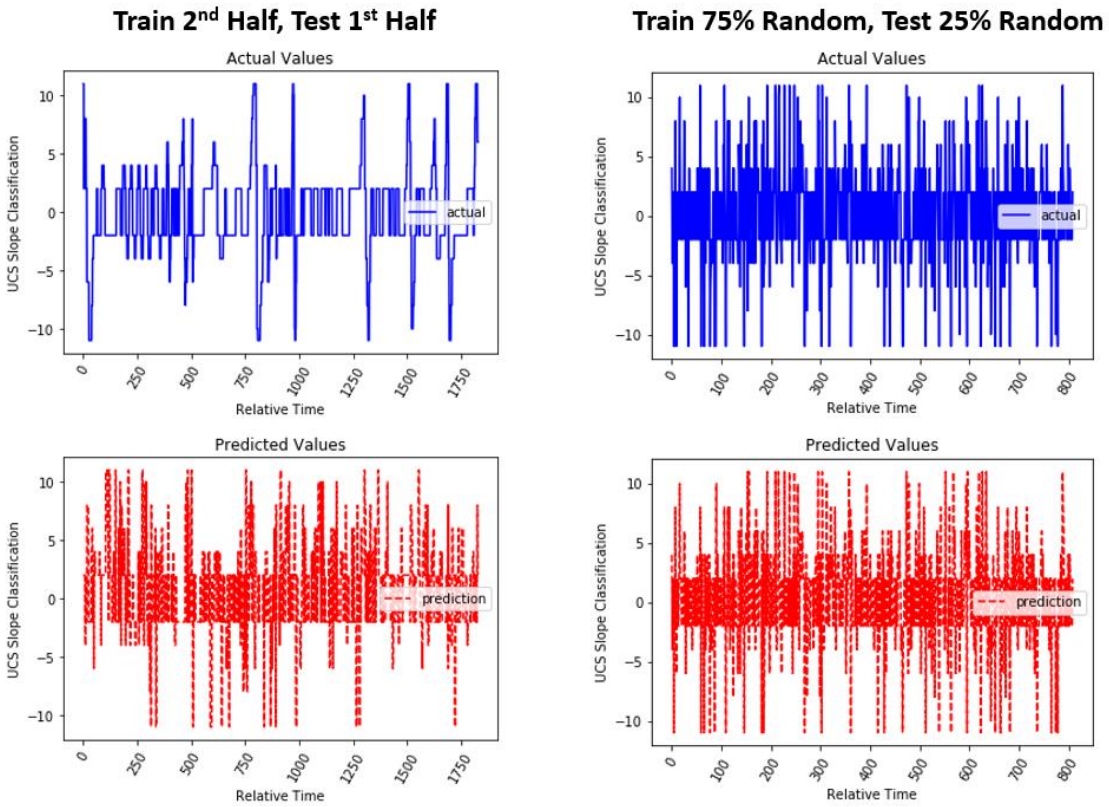


Figure 4.18: Actual and Predicted UCS Slope Classification vs. Relative Time

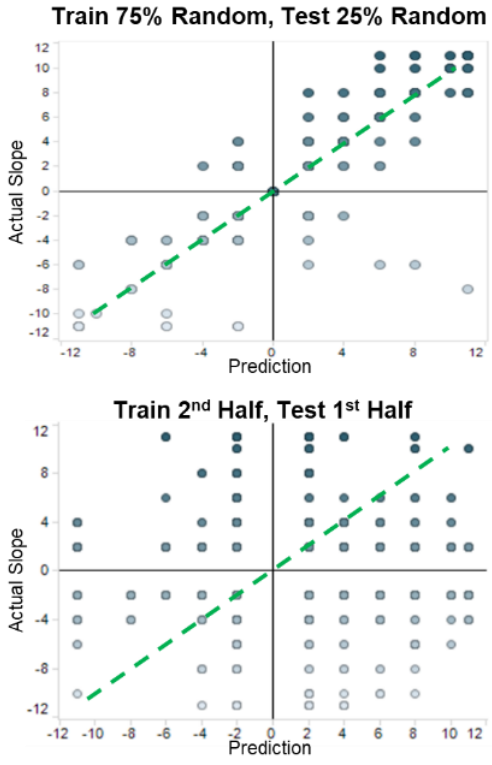


Figure 4.19: Actual vs. Predicted UCS Slope Classification

The results in Figure 4.19 show a stronger prediction accuracy for the randomly selected 75/25 split train and test set analysis. The absolute accuracy of each case was 28% for the half and half split approach and 84% for the randomly selected train and test set approach. Better accuracy with the randomly selected train and test set approach is expected because the random selection of data points increases the likelihood of close or consecutive data points to be learned from and then tested on. When the trends are based on the parameter trends, this means the likelihood that the algorithm has learned from a similar or the same feature combination is high.

The Well 8 UCS data was obtained in a vertical pilot hole covering primarily the curve and lateral sections of the wellbore. To reduce the extent of lateral extrapolation of

the vertically obtained UCS data when mapping it along the deviated wellbore, the analysis was confined to the curve section of the well. Therefore, when the data is split in half based on MD, the algorithm learns from the second half or most deviated part of the curve. When applying these learnings to the first half or most vertical section of the curve, the drilling state may have changed such that the specific parameter states learned in the more deviated section no longer indicate the same UCS change. As seen in Figure 4.7 in Section 4.1.2, the parameter behavior can change when the drilling state changes, even from stand to stand, or a variable in the drilling context changes. The change in inclination may influence this in the case of the curve.

Alternatively, the lower accuracy may be due to a higher frequency of DE influence in the first section of the curve than the second section. If DE changes tend to dominate the parameter responses in the test set, and UCS changes tend to dominate in the training set, the UCS change based learnings will be inaccurate when applied to the test set, if the features as defined are sufficient to distinguish between a UCS and DE influence. The data labels provided in both the train and test sets are applied directly from the UCS data, and do not include any defined instance of DE change. Therefore, the training set in either case likely has instances of DE influence that acts as noise in the models. If we see repeated feature combinations in the 1st half versus 2nd half of the data with a high degree of accuracy in their UCS slope prediction, the likelihood that that parameter behavior definition describes a specific behavior, either due to UCS or DE influence, may be high. A similar conclusion may be drawn from the random selection method, however the prediction results on the test set may be biased. In either case, there is potential for a UCS slope to be incorrectly learned where DE is actually taking place. Therefore, a manual review of where the classified parameter behaviors occur in both the train and test sets should be used to

validate whether the assigned UCS change or a potential DE is dominating. Additional data would be required to build a library of definitions of parameter behaviors for indication of UCS and DE influences across a range of drilling contexts.

Inaccuracy in the prediction may also indicate that the description of parameter behavior by the currently defined features may not be sufficient to distinguish between a UCS and a DE change. To enhance this, the features can be modified from their current definition and additional features can be added to improve the model results. Adding features which describe specific drilling states, such as inclination, tortuosity, bit hydraulics, or other system changes add additional context to a specific parameter behavior that can be used in building the prediction model.

Currently, the features contributing the most to the accuracy of the model in both test/train cases are the linear regression slopes. Therefore, the high-level parameter trends may be better for identifying UCS changes in the parameter responses. This again suggests a transition state in the parameter data in response to a UCS change. This may be due to a delay in the measured response at surface to a downhole state change, making downhole parameter estimation or measurement a possible way to improve the model accuracy.

Based on the results of the initial implementation, the Random Forest model shows some promise in defining parameter responses specific to UCS and DE influence. The data set used in this analysis was limited in both size and diversity of drilling context. The majority of the curve is drilled via slide drilling, therefore the baseline DE is likely low over this entire section. A more significant UCS change may be required to register a response in the data due to the constant high level of DE influence it must compete with. With additional drilling data, and associated UCS data, a more robust Random Forest prediction model is possible. Through manual validation of the results, behavior categories

may be defined at a scale of interest for qualitative interpretation of UCS influence along the wellbore. Where relative DE influences are dominating, real time recognition based on the parameter response may also be possible with potential drilling optimization benefits.

Chapter 5: Conclusion

MSE, driven by the drilling parameters, will exhibit changes as the parameters respond to system energy influences, either from a change in rock strength or the efficiency state of the system. Dupriest and Koederitz (2005) have repeatedly shown that changes in MSE can be interpreted as relative changes in the DE state of the system when UCS, or rock strength is assumed constant. Conversely, Logan (2015) assumes a constant DE over the length of a completion stage and interprets MSE changes as instances of changes in rock strength. At a high level, either case may be adequate for the application intended, particularly in the case of drilling optimization where a large, unexpected MSE increase is most likely due to an inefficiency the driller can respond to. However, as the resolution of interpretation required for a given application increases, the assumptions begin to break down due to a constant and dynamic level of efficiency as the bit drills ahead. Over a depth of interest for completions design, the assumption of a constant DE may not be realistic. Therefore, in order to interpret rock strength changes directly from the MSE term, first DE must be optimized to prevent high frequency changes in DE that will mute any rock strength response. Second, a DE change must be distinguishable from a rock strength change up to some resolution of interest. As shown in the work presented in this thesis, MSE cannot distinguish between two different types of energy influences. Therefore, the parameters themselves tell us more about what is happening downhole than the MSE term.

Both manual and automated pattern recognition methods were implemented in this work in an attempt to identify unique parameter behaviors associated with a specific energy change. While evidence suggests that the parameters respond differently to a change in rock strength, DE, or combination of the two, defining the nature of that behavior proved challenging. A particular definition of parameter behavior must be detailed enough to

capture the initial change, the duration over which it is changing, and the stabilization period after the change has taken place. In addition, it must be designed as a codified fingerprint to be used in an automated detection tool to allow for recognition of the behavior in real time and reduce the need for subjective interpretation. The features designed in the Random Forest algorithm presented here appear to capture some of the detail necessary to achieve these goals, particularly at a high level, however future work is recommended to improve the descriptive features. Additional wellbore and drill string models may be introduced to estimate torque and drag, confining pressure at the bit, and bit wear to help validate potential instances of DE influence. Downhole parameter estimation or measurement will likely improve the accuracy of the descriptions and thus the prediction model. Ultimately, manual review will be required to validate the model results as the features are redesigned and enhanced using a combination of these tools. Random Forest was the chosen Machine Learning model used in this analysis but other Machine Learning approaches may prove more efficient for development and effective in their predictions. Therefore, future work should not be limited to the Random Forest model presented here.

Future work will require more drilling data with accompanying UCS data preferably taken from the same well. This combination of data is not as readily available as drilling data alone, therefore some extrapolation of offset UCS data may be necessary. Prioritization of the minimum offset UCS data is recommended to reduce the risk of TVD inaccuracies. In addition, the continued development of the Random Forest, or other Machine Learning models will benefit initially from data taken within similar drilling contexts to further prove out the concept. The learnings from this data set will be robust within the given drilling context and provide a good learnings foundation where new

contextual variables can be introduced. Ultimately, the model will benefit from a large data set with diverse drilling contexts to build a library of behaviors for predictions. If this library of learned behaviors can accurately distinguish a rock strength change, the location and nature of that change can be understood without the need for costly and time-consuming logging operations, and would be available to inform completion designs as soon as the driller reaches total depth (TD).

Appendix

A. Processing Well Data: Master Program

The program presented below is a generic template that can be used to process the well data provided for any of the 10 wells in the data set analyzed in this work. Some modification would be required to implement this program, and the associated programs presented in the following appendices, on a new data set as specific data header titles, parameter units, and other data source specific details will vary from data set to data set. Each well is referred to as "Pad X", where X should be replaced with the associated well number from the data set analyzed in this work.

```
%Process well data set from various data sources

%%Set smoothing and binning parameters
x=input('Input moving average interval= ');
stepsize=input('Input Bin Length = ');

%%Import drilling data headers
opts=detectImportOptions('Pad X.csv');
opts.SelectedVariableNames ={'HoleDepth' 'WeightOnBit' 'BitRPM'
'DifferentialPressure' 'RateOfPenetration' 'MotorRPM' 'BlockHeight' 'RotaryRPM'
'TotalPumpOutput' 'BitDepth' 'RotaryTorque' 'Inclination' 'Azimuth'}; %Can use
header indeces here instead of names
A=readtable('Pad X.csv',opts);
MSEdata=table2array(A);
clear A

%%Import proprietary drilling dysfunction indicator data
opts.SelectedVariableNames ={'Depth_ft_' 'BitBallingBelief' 'BitBounceBelief'
'StickSlipBelief' 'WhirlBelief' 'OtherDrillingDysfunctionBelief'
'NoDrillingDysfunctionBelief' 'UCS_Ids'};
A=readtable('Pad X.csv',opts);
DysfunctionandUCSData=table2array(A);
clear A

%%Combine imported data sets
MSEdata=[MSEdata DysfunctionandUCSData];
```

```
%%Import BHA and drill string data
opts=detectImportOptions('Pad X MSE.csv');
index=[3 5 6 7 8 9]; %[BHA# DepthIn BitSize RevPerGal Tmax dPmax]
opts.SelectedVariableNames=index;
A=readtable('Pad C MSE.csv',opts);
BHAinfo=table2array(A);
BHAinfo2=BHAinfo;
clear A

%%Import fluid properties
opts=detectImportOptions('Pad X MR.csv');
index=[3 4 10 65];
opts.SelectedVariableNames =index;
A=readtable('Pad 8 MR.csv',opts);
HSIinfo=table2array(A(:,1:4));
clear A

%%Import survey data
A=readtable('PAD_8_Depth_Reference_Data.csv');
Survey=table2array(A);
clear A

%%Run data processing programs (see Appendices below)
run('BHAdataprocess.m')
run('MSEdataprocess.m')
run('CalculateDrillingMetrics.m')
run('SurveyMap.m')

%%Import Pad 8 UCS data
opts=detectImportOptions('Pad 8.csv');
opts.SelectedVariableNames={'UCS_i'};
A=readtable('Pad 8.csv',opts);
UCSpad8=table2array(A);
clear A
MSEdata=[MSEdata UCSpad8];

%%Clean compiled data set
run('SlidingClean.m')

%%Extract Pad 8 UCS to move location of data header within data set (not
%%necessary)
Pad8UCS=MSEdataclean(:,35);
MSEdataclean(:,35)=[];

%%Smooth MSE using moving average
```

```
smoothPad8MSE=smooth(MSEdataclean(:,23),x);
MSEdataclean=[MSEdataclean smoothPad8MSE];

%%Run analytics programs (see below Appendices)
run('Derivatives.m')
run('ChangeinRockStrength.m')
run('firstorderparametercombos.m')
run('MoreMetrics.m')

MSEdataclean=[MSEdataclean smoothparam Pad8UCS]; %Pad 8 UCS data will always be
at end of master file

%%Create .cvs file of cleaned, processed master data set
PadX=MSEdataclean;
PadXMSE=array2table(MSEdataclean);
PadXMSE.Properties.VariableNames={'HoleDepth' 'WeightonBit' 'BitRPM'
'DifferentialPressure' 'RateofPenetration' 'MotorRPM' 'BlockHeight' 'RotaryRPM'
'TotalPumpOutput' 'BitDepth' 'RotaryTorque' 'Inclination' 'Azimuth'
'DysfunctionDepth' 'BitBallingBelief' 'BitBounceBelief' 'StickSlipBelief'
'WhirlBelief' 'OtherDysfunctionBelief' 'NoDysfunctionBelief' 'UCSanalog' 'BHA'
'MSE' 'MSEwithRotaryTorque' 'Time' 'CalcedBitRPM' 'DrillingEfficiency'
'BitAggressiveness' 'HSI' 'Torque' 'TVDSurvey' 'CourseNorthSurvey'
'CourseEastSurvey' 'TVD' 'CourseNorth' 'CourseEast' 'BinnedMSE' 'SmoothMSE'
'dROP' 'dRPM' 'dTorque' 'dWOB' 'dTime' 'dsmoothROP' 'dsmoothRPM'
'dsmoothTorque' 'dsmoothWOB' 'dsmoothTime' 'sWOBandTvsROP' 'ZeroRatio'
'MicroErraticTorque' 'MicroErraticWOB' 'MicroErraticRPM' 'MicroErraticROP'
'TorqueMagnitudeChange' 'WOBMagnitudeChange' 'RPMMagnitudeChange'
'ROPMagnitudeChange' 'sWOBandTandROP' 'sWOBandROPvsT' 'sTandROPvsWOB'
'WOBandTandROP' 'WOBandROPvsT' 'TandROPvsWOB' 'WOBandTvsROP' 'ROPandWOB'
'CoefficientofFriction' 'dROPvdRPM' 'dROPvdT' 'DOC' 'smoothTorque' 'smoothROP'
'smoothWOB' 'smoothRPM' 'smoothTime' 'Pad8UCS'}; %Ensure all headers have a
correctly defined title
writetable(PadXMSE,'C:\Users\Carolyn\Documents\Masters Research\Spring
2018\Apache\Processed Single Files\PadXMSETotal.csv') %File location to be
specific to user computer

clear i smoothparam stepsize x smoothPad8MSE index DysfunctionandUCSData opts
BHAinfo MSEdata HSIinfo Survey UCSpad8 Pad8UCS
```

B. PROCESS BHA DATA

The program presented below is referred to in the master program in Appendix A as “BHAdataprocess.m”.

```
%Process BHA info

%%Index Bit Depth In greater than 0
iBHAinfo=find(BHAinfo(:,2)>0); %DepthIn greater than 0
BHAinfo=BHAinfo(iBHAinfo,:);

%%Create vectors of equal length to master data set of BHA data
n=numel(BHAinfo(:,1));
m=numel(MSEdata(:,1));
BHAnum=zeros(m,1);
revpgal=zeros(m,1);
BitSize=zeros(m,1);
Tmax=zeros(m,1);
dPmax=zeros(m,1);
for i=1:n-1
    idepth=find(MSEdata(:,1)<=BHAinfo(i+1,2) & MSEdata(:,1)>=BHAinfo(i,2));
    BHAnum(idepth,1)=BHAinfo(i,1);
    revpgal(idepth,1)=BHAinfo(i,4);
    BitSize(idepth,1)=BHAinfo(i,3);
    Tmax(idepth,1)=BHAinfo(i,5);
    dPmax(idepth,1)=BHAinfo(i,6);
end
idepth=find(MSEdata(:,1)>BHAinfo(n,2));
BHAnum(idepth,1)=BHAinfo(n,1);
revpgal(idepth,1)=BHAinfo(n,4);
BitSize(idepth,1)=BHAinfo(n,3);
Tmax(idepth,1)=BHAinfo(n,5);
dPmax(idepth,1)=BHAinfo(n,6);

MSEdata=[MSEdata BHAnum];

clear iBHAinfo
clear n
clear m
clear idepth
```

C. CALCULATE MSE

The program presented below is referred to in the master program in Appendix A as “MSEdataprocess.m”.

```
%%Calculate MSE
```

```
%Calculate MSE using calculated torque
MSEcalced=((4*MSEdata(:,2)*1000)./(pi*(BitSize(:,1).^2)))+(((480*(MSEdata(:,8)+
(MSEdata(:,9).*revpgal(:,1)))).*((Tmax(:,1)./dPmax(:,1)).*MSEdata(:,4))).//((Bit
Size(:,1).^2).*MSEdata(:,5)));
MSEdata=[MSEdata MSEcalced];

%Calculate MSE using measured torque (provided in raw drilling data)
MSEcalcedwT=((4*MSEdata(:,2)*1000)./(pi*(BitSize(:,1).^2)))+(((480*(MSEdata(:,8
)+(MSEdata(:,9).*revpgal(:,1)))).*(MSEdata(:,11)./((BitSize(:,1).^2).*MSEdata(:
,5)));
MSEdata=[MSEdata MSEcalcedwT];

clear MSEcalcedwT MSEcalced

%%Create Cumulative Time vector
n=numel(MSEdata(:,1));
Time10sec=[1:n]'; %data points taken every 10 seconds
Timemin=Time10sec/6; %convert to cumulative min
Time=Timemin/60; %convert to cumulative hrs
clear Timemin
clear Time10sec
MSEdata=[MSEdata Time];

clear Time

%%Create calculated bit RPM vector
CalcedBitRPM=MSEdata(:,8)+(MSEdata(:,9).*revpgal(:,1));
MSEdata=[MSEdata CalcedBitRPM];

clear CalcedBitRPM

%%Calculate DE using offset UCS data
DE=MSEdata(:,21)./MSEdata(:,23); %Drilling Efficiency=UCS/MSE
MSEdata=[MSEdata DE];

clear DE n
```

D. CALCULATE DRILLING METRICS

The program presented below is referred to in the master program in Appendix A as “CalculateDrillingMetrics.m”.

```
%%Calculate Bit Aggressiveness (BA)
T=((Tmax(:,1)./dPmax(:,1)).*MSEdata(:,4)); %Torque calculation (same as used in
MSE with Rotary Torque) - NEED TO CHANGE BACK
BA=(36*T(:,1))./((MSEdata(:,2)*1000).*BitSize(:,1));

%%Calculate HSI

%Create Mud Weight vector
MWsubset=HSIinfo;
i=isnan(MWsubset(:,4));
MWsubset(i,:)=[];
n=numel(MWsubset(:,1));
m=numel(MSEdata(:,1));
MudWeight=zeros(m,1);
for i=1:n
    idepth=find(MSEdata(:,1)<=MWsubset(i,2) & MSEdata(:,1)>=MWsubset(i,1));
    MudWeight(idepth,1)=MWsubset(i,4);
end
clear MWsubset
clear idepth

%Create TFA (total flow area) vector
TFAsubset=HSIinfo;
i=isnan(TFAsubset(:,3));
TFAsubset(i,:)=[];
n=numel(TFAsubset(:,1));
TFA=zeros(m,1);
for i=1:n
    iTFA=find(MSEdata(:,1)<=TFAsubset(i,2) & MSEdata(:,1)>=TFAsubset(i,1));
    TFA(iTFA,1)=TFAsubset(i,3);
end
clear TFAsubset
clear iTFA

%Calculate HSI (using "total pump output" for Q)
dPbit=((((8.311e-
5)*MudWeight(:,1))).*(MSEdata(:,9).^2))./((0.95^2)*(TFA(:,1).^2));
HPbit=(dPbit.*MSEdata(:,9))/1714;
HSI=(1.273*HPbit)./(BitSize(:,1).^2);
n=numel(HSI);
for i=1:n %forces max to be 11
    if HSI(i,1)>11
        HSI(i,1)=11;
    else
        HSI(i,1)=HSI(i,1);
```

```
    end
end
clear dPbit
clear HPbit
clear n
clear i
clear m
%%Add Metrics to master data set
MSEdata=[MSEdata BA HSI T];

clear BA HSI T TFA MudWeight BHAnum revpgal BitSize Tmax dPmax
```

E. MAP EXTERNAL SURVEY DATA

The program presented below is referred to in the master program in Appendix A as “SurveyMap.m”.

```
%Map Survey Data (from separate file) to master data set
n=numel(Survey(:,1));
m=numel(MSEdata(:,1));

Azimuth=Survey(:,3)*(pi/180); %Azimuth angle in radians
Inclination=Survey(:,2)*(pi/180); %Inclination in radians

MDsegment=diff(Survey(:,1));
MDsegment=[0; MDsegment];

TVD=zeros(n,1);
TVD(1,1)=Survey(1,1);
TVDsegment=zeros(n,1);
avginclination=zeros(n,1);
avgazimuth=zeros(n,1);
for i=2:n
  avginclination(i,1)=(Inclination(i,1)+Inclination(i-1,1))/2;
  avgazimuth(i,1)=(Azimuth(i,1)+Azimuth(i-1,1))/2;
  TVDsegment(i,1)=MDsegment(i,1)*cos(avginclination(i,1));
  TVD(i,1)=TVD(i-1,1)+TVDsegment(i,1);
end

courseNorth=zeros(n,1);
courseEast=zeros(n,1);
coordNorth=zeros(n,1);
coordEast=zeros(n,1);
for i=2:n
```

```
courseNorth(i,1)=MDsegment(i,1).*sin(avginclination(i,1)).*cos(avgazimuth(i,1))
;

courseEast(i,1)=MDsegment(i,1).*sin(avginclination(i,1)).*sin(avgazimuth(i,1));
    coordNorth(i,1)=coordNorth(i-1,1)+courseNorth(i,1);
    coordEast(i,1)=coordEast(i-1,1)+courseEast(i,1);
end

TVDmap=zeros(m,1);
Northmap=zeros(m,1);
Eastmap=zeros(m,1);

for i=2:n
    lindex=find(MSEdata(:,1)<=Survey(i,1) & MSEdata(:,1)>=Survey(i-1,1));
    TVDmap(lindex,1)=TVD(i,1);
    Northmap(lindex,1)=coordNorth(i,1);
    Eastmap(lindex,1)=coordEast(i,1);
end

tailend=find(MSEdata(:,1)>Survey(end,1));
TVDmap(tailend,1)=TVD(end,1);
Northmap(tailend,1)=coordNorth(end,1);
Eastmap(tailend,1)=coordEast(end,1);


MSEdata=[MSEdata TVDmap Northmap Eastmap];

clear m n tailend TVDmap Northmap Eastmap lindex courseNorth courseEast
coordNorth coordEast Azimuth Inclination MDsegment TVD TVDsegment
```

F. CLEAN THE RAW DATA SET

The program presented below is referred to in the master program in Appendix A as “SlidingClean.m”. The programs called out in the below are presented in the following Appendices.

```
%Clean the raw master data set

MSEdataclean=MSEdata;

%%Remove upward movement of traveling block
dH=diff(MSEdataclean(:,7));
```

```
IdH=find(dH<0);
MSEdataclean=MSEdataclean(IdH,:);

%%Remove ROP=0 or neg or less than 40 ft/hr
IROP=find(MSEdataclean(:,5)>0);
MSEdataclean=MSEdataclean(IROP,:);

%%Remove dP= neg or 0
IdP=find(MSEdataclean(:,4)>0);
MSEdataclean=MSEdataclean(IdP,:);

%%Remove hole depth not changing
ddepth=diff(MSEdataclean(:,1));
Iddepth=find(ddepth>0);
MSEdataclean=MSEdataclean(Iddepth,:);

%%Remove bit position change =0 or negative
dbit=diff(MSEdataclean(:,10));
Idbit=find(dbit>0);
MSEdataclean=MSEdataclean(Idbit,:);

%%Remove Mud Motor RPM = zero (NOT surface RPM =0), if provided
IMMRPM=find(MSEdataclean(:,6)>0);
MSEdataclean=MSEdataclean(IMMRPM,:);

%%Remove WOB <=0
iWOB=find(MSEdataclean(:,2)>0);
MSEdataclean=MSEdataclean(iWOB,:);

%%Smooth Flow Rate (Q=Total Pump Out) to remove spikes
iQ=find(MSEdataclean(:,9)<=1000);
MSEdataclean=MSEdataclean(iQ,:);

%%Map to TVD
run('MapTVD.m')

%%Bin MSE
run('MSEbinning.m')

%%Remove Sliding and RPM downward spikes - IF NECESSARY
isliding=find(MSEdataclean(:,8)>60); %where RotaryRPM>60
MSEdataclean=MSEdataclean(isliding,:);

clear isliding
clear iWOB iQ
clear IMMRPM
```

```
clear dbit
clear Idbit
clear ddepth
clear Iddepth
clear IROP
clear dH
clear IdH
clear IdP
```

G. MAP TVD

The program presented below maps the survey data provided in the raw drilling data to generate a high resolution TVD, and is referred to in the cleaning program in Appendix F as “MapTVD.m”.

```
%Map MD to TVD using survey data in master file (not raw additional survey
%data provided)
n=numel(MSEdataclean(:,1));
Azimuth=MSEdataclean(:,13)*(pi/180); %Azimuth angle in radians
Inclination=MSEdataclean(:,12)*(pi/180); %Inclination in radians

MDsegment=diff(MSEdataclean(:,1));
MDsegment=[0; MDsegment];

TVD=zeros(n,1);
TVD(1,1)=MSEdataclean(1,1);
TVDsegment=zeros(n,1);
avginclination=zeros(n,1);
avgazimuth=zeros(n,1);
for i=2:n
  avginclination(i,1)=(Inclination(i,1)+Inclination(i-1,1))/2;
  avgazimuth(i,1)=(Azimuth(i,1)+Azimuth(i-1,1))/2;
  TVDsegment(i,1)=MDsegment(i,1)*cos(avginclination(i,1));
  TVD(i,1)=TVD(i-1,1)+TVDsegment(i,1);
end

MSEdataclean=[MSEdataclean TVD];

courseNorth=zeros(n,1);
courseEast=zeros(n,1);
coordNorth=zeros(n,1);
coordEast=zeros(n,1);
for i=2:n
```

```
courseNorth(i,1)=MDsegment(i,1).*sin(avginclination(i,1)).*cos(avgazimuth(i,1))
;

courseEast(i,1)=MDsegment(i,1).*sin(avginclination(i,1)).*sin(avgazimuth(i,1));
    coordNorth(i,1)=coordNorth(i-1,1)+courseNorth(i,1);
    coordEast(i,1)=coordEast(i-1,1)+courseEast(i,1);
end

MSEdataclean=[MSEdataclean coordNorth coordEast];

clear Azimuth
clear Inclination
clear MDsegment
clear TVD
clear TVDsegment
clear avginclination
clear avgazimuth
clear courseNorth
clear courseEast
clear coordNorth
clear coordEast
clear n
```

H. BIN MSE

The program presented below is referred to in the cleaning program in Appendix F as “MSEbinning.m”.

```
%Split MSE output into bins and take average over each bin

%%Remove depths captured prior to depth in of first BHA
mindepth=BHAinfo(1,2);
indexmindepth=find(MSEdataclean(:,1)>=mindepth);
MSEdataclean=MSEdataclean(indexmindepth,:);

%%Bin the MSE values from depth in of first BHA
mindepth=min(MSEdataclean(:,1))+500;
maxdepth=max(MSEdataclean(:,1));
j=maxdepth/stepsize; %approximate # of groups needed
j1=ceil(j);%rounds up to nearest integer
n=numel(MSEdataclean(:,1));
groups=ones(n,1);
for i=1:j1
```

```
    int1=mindepth+(i*stepsize);
    int2=mindepth+((i-1)*stepsize);
    depthinterval=find(MSEdataclean(:,1)>=int2 & MSEdataclean(:,1)<int1);
    groups(depthinterval,1)=i; %creates [1 1 2 2 3 3..]' vector indicating
groups
end
binMSE=splitapply(@mean,MSEdataclean(:,23),groups);
m=numel(binMSE(:,1));
extendbinMSE=ones(n,1);
for i=1:m
    index=find(groups(:,1)==i);
    extendbinMSE(index,1)=binMSE(i,1);
end

MSEdataclean=[MSEdataclean extendbinMSE];

clear mindepth
clear indexmindepth
clear extendbinMSE
clear index
clear binMSE
clear m
clear n
clear i
clear int1
clear int2
clear depthinterval
clear groups
clear maxdepth
clear j
clear j1
```

I. Smooth Parameters and Calculate Derivatives

The program presented below is referred to in the master program in Appendix A as “Derivatives.m”.

```
%Smooth parameters and take derivatives

dROP=diff(MSEdataclean(:,5));
dROP=[0;dROP];
smoothROP=smooth(MSEdataclean(:,5),x);
dsmoothROP=diff(smoothROP);
dsmoothROP=[0;dsmoothROP];
```

```
RPM=MSEdataclean(:,8)+MSEdataclean(:,26);
dRPM=diff(RPM);
dRPM=[0;dRPM];
smoothRPM=smooth(RPM,x);
dsmoothRPM=diff(smoothRPM);
dsmoothRPM=[0;dsmoothRPM];
dTorque=diff(MSEdataclean(:,31));
dTorque=[0;dTorque];
smoothTorque=smooth(MSEdataclean(:,31),x);
dsmoothTorque=diff(smoothTorque);
dsmoothTorque=[0;dsmoothTorque];
dWOB=diff(MSEdataclean(:,2));
dWOB=[0;dWOB];
smoothWOB=smooth(MSEdataclean(:,2),x);
dsmoothWOB=diff(smoothWOB);
dsmoothWOB=[0;dsmoothWOB];
dTime=diff(MSEdataclean(:,25));
dTime=[0;dTime];
smoothTime=smooth(MSEdataclean(:,25),x);
dsmoothTime=diff(smoothTime);
dsmoothTime=[0;dsmoothTime];
%MD=MSEdataclean(:,1);
%TVD=MSEdataclean(:,35);
%dRPMvdROP=dRPM./dROP;
%dTvdROP=dTorque./dROP;
%dRPMvdT=dRPM./dTorque;
%dWOBvdROP=dWOB./dROP;



Deriv=[dROP dRPM dTorque dWOB dTime dsmoothROP dsmoothRPM dsmoothTorque
dsmoothWOB dsmoothTime];
smoothparam=[smoothTorque smoothROP smoothWOB smoothRPM smoothTime];
clear dROP RPM dRPM dTorque dWOB dTime smoothROP smoothRPM smoothTorque
smoothWOB smoothTime dsmoothROP dsmoothRPM dsmoothTorque dsmoothWOB dsmoothTime
MSEdataclean=[MSEdataclean Deriv];
clear Deriv
```

J. MANUAL PARAMETER BEHAVIOR CLASSIFICATION

The program presented below is referred to in the master program in Appendix A as “ChangeinRockStrength.m”.

```
%Rock Strength Change Classification Code
```

```
%WOB, Torque go UP, ROP goes DOWN = increase in rock strength
%WOB, Torque go DOWN, ROP goes UP = decrease in rock strength
%Using derivative variables

n=numel(MSEdataclean(:,1));
dsmoothWOB=MSEdataclean(:,48);
dsmoothTorque=MSEdataclean(:,47);
dsmoothROP=MSEdataclean(:,45);
dsmoothRPM=MSEdataclean(:,46);
sWOBandTvsROP=zeros(n,1);
for i=1:n
if dsmoothWOB(i)>0 && dsmoothTorque(i)>0 && dsmoothROP(i)<0
    sWOBandTvsROP(i)=1;
end
if dsmoothWOB(i)<0 && dsmoothTorque(i)<0 && dsmoothROP(i)>0
    sWOBandTvsROP(i)=-1;
end
end

MSEdataclean=[MSEdataclean sWOBandTvsROP];
%%Smooth out the rock strength change indicator and indicate level of
%eratic behavior

%Classify eratic behavoir
n=numel(MSEdataclean(:,1));
span=50; %span of microinterval
dTorque=MSEdataclean(:,42);
absdTorque=abs(dTorque);
Torquemagnitudechange=zeros(n,1);
for i=1:n
    if absdTorque(i)>10000
        Torquemagnitudechange(i)=7;
    elseif absdTorque(i)<=10000 && absdTorque(i)>7000
        Torquemagnitudechange(i)=6;
    elseif absdTorque(i)<=7000 && absdTorque(i)>5000
        Torquemagnitudechange(i)=5;
    elseif absdTorque(i)<=5000 && absdTorque(i)>3000
        Torquemagnitudechange(i)=4;
    elseif absdTorque(i)<=3000 && absdTorque(i)>1000
        Torquemagnitudechange(i)=3;
    elseif absdTorque(i)<=1000 && absdTorque(i)>500
        Torquemagnitudechange(i)=2;
    elseif absdTorque(i)<=500
        Torquemagnitudechange(i)=1;
    end
end
```

```
Torqueindicator=zeros(n,1);
for i=1:n %converts dTorque to categories (pos, neg, zero)
    if dTorque(i)<0
        Torqueindicator(i)=-1;
    elseif dTorque(i)>0
        Torqueindicator(i)=1;
    else
        Torqueindicator(i)=0;
    end
end

Zeroratio=zeros(n,1);
for i=1+span:n
    microinterval= Torqueindicator(i-span:i,1);
    m=numel(microinterval);
    pos=find(microinterval>0);
    neg=find(microinterval<0);
    zero=find(microinterval==0);
    P=numel(pos);
    N=numel(neg);
    Z=numel(zero);
    Pratio=P/m;
    Nratio=N/m;
    Zratio=Z/m;
     if Zratio<=.25
         Zeroratio(i)=1; %highly unstable
     elseif Zratio>.25 && Zratio<=.5
         Zeroratio(i)=2; %moderately unstable
     elseif Zratio>.5 && Zratio<=.75
         Zeroratio(i)=3; %moderately stable
     else
         Zeroratio(i)=4; %highly stable
     end
end

MSEdataclean=[MSEdataclean Zeroratio];

EraticTorqueIndicator=zeros(n,1);
for i=1:n-1
    if Torqueindicator(i)-Torqueindicator(i+1)==2 || Torqueindicator(i)-
Torqueindicator(i+1)==-2
        EraticTorqueIndicator(i+1)=2; %oscillating between two consecutive
points
    elseif Torqueindicator(i)-Torqueindicator(i+1)==0
```

```
        EraticTorqueIndicator(i+1)=1; %continuing to increase or decrease
between two consecutive points
    elseif Torqueindicator(i)-Torqueindicator(i+1)==1 || Torqueindicator(i)-
Torqueindicator(i+1)==-1
        EraticTorqueIndicator(i+1)=0; %stabilizing or plateauing between two
consecutive points
    end
end

MicroEraticTorque=zeros(n,1);
for i=1+span:n
    microinterval2= EraticTorqueIndicator(i-span:i,1); %last 50 data points
prior to current of dTorque
    m=numel(microinterval2);
    osc=find(microinterval2==2);
    cont=find(microinterval2==1);
    stab=find(microinterval2==0);
    O=numel(osc);
    C=numel(cont);
    S=numel(stab);
    Oratio=O/m;
    if Oratio==0
        MicroEraticTorque(i)=0; %stable
    elseif Oratio>0 && Oratio<=.25
        MicroEraticTorque(i)=1; %fairly stable
    elseif Oratio>.25 && Oratio<=.5
        MicroEraticTorque(i)=2; %somewhat oscillating
    elseif Oratio>.5 && Oratio<=.75
        MicroEraticTorque(i)=3; %moderately oscillating
    elseif Oratio>.75 && Oratio<1
        MicroEraticTorque(i)=4; %highly oscillating
    elseif Oratio==1
        MicroEraticTorque(i)=5; %fully oscillating
    end
end

MSEdataclean=[MSEdataclean MicroEraticTorque];

clear dTorque microinterval microinterval2 i m osc cont stab O C S Oratio pos
neg zero P N Z Pratio Nratio Zratio

%%Define WOB eratic behavior
dWOB=MSEdataclean(:,43);
absdWOB=abs(dWOB);
WOBmagnitudechange=zeros(n,1);
for i=1:n
```

```
    if absdWOB(i)>30
        WOBmagnitudechange(i)=6;
    elseif absdWOB(i)<=30 && absdWOB(i)>20
        WOBmagnitudechange(i)=5;
    elseif absdWOB(i)<=20 && absdWOB(i)>15
        WOBmagnitudechange(i)=4;
    elseif absdWOB(i)<=15 && absdWOB(i)>10
        WOBmagnitudechange(i)=3;
    elseif absdWOB(i)<=10 && absdWOB(i)>5
        WOBmagnitudechange(i)=2;
    elseif absdWOB(i)<=5
        WOBmagnitudechange(i)=1;
    end
end

dWOBindicator=zeros(n,1);
for i=1:n %converts dTorque to categories (pos, neg, zero)
    if dWOB(i)<0
        dWOBindicator(i)=-1;
    elseif dWOB(i)>0
        dWOBindicator(i)=1;
    else
        dWOBindicator(i)=0;
    end
end

EraticWOBIndicator=zeros(n,1);
for i=1:n-1
    if dWOBindicator(i)-dWOBindicator(i+1)==2 || dWOBindicator(i)-
dWOBindicator(i+1)==-2
        EraticWOBIndicator(i+1)=2; %oscillating between two consecutive points
    elseif dWOBindicator(i)-dWOBindicator(i+1)==0
        EraticWOBIndicator(i+1)=1; %continuing to increase or decrease between
two consecutive points
    elseif dWOBindicator(i)-dWOBindicator(i+1)==1 || dWOBindicator(i)-
dWOBindicator(i+1)==-1
        EraticWOBIndicator(i+1)=0; %stabilizing or plateauing between two
consecutive points
    end
end

MicroEraticWOB=zeros(n,1);
for i=1+span:n
    microinterval3= EraticWOBIndicator(i-span:i,1); %last 50 data points prior
to current of dTorque
    m=numel(microinterval3);
```

```
    osc=find(microinterval3==2);
    cont=find(microinterval3==1);
    stab=find(microinterval3==0);
    O=numel(osc);
    C=numel(cont);
    S=numel(stab);
    Oratio=O/m;
    if Oratio==0
        MicroEraticWOB(i)=0; %stable
    elseif Oratio>0 && Oratio<=.25
        MicroEraticWOB(i)=1; %fairly stable
    elseif Oratio>.25 && Oratio<=.5
        MicroEraticWOB(i)=2; %somewhat oscillating
    elseif Oratio>.5 && Oratio<=.75
        MicroEraticWOB(i)=3; %moderately oscillating
    elseif Oratio>.75 && Oratio<1
        MicroEraticWOB(i)=4; %highly oscillating
    elseif Oratio==1
        MicroEraticWOB(i)=5; %fully oscillating
    end
end


MSEdataclean=[MSEdataclean MicroEraticWOB];

clear dWOB microinterval3 i m osc cont stab O C S Oratio MicroEraticWOB
EraticWOBIndicator dWOBindicator

%%Define eratic RPM behavior
dRPM=MSEdataclean(:,41);
absdRPM=abs(dRPM);
RPMmagnitudechange=zeros(n,1);
for i=1:n
    if absdRPM(i)>40
        RPMmagnitudechange(i)=6;
    elseif absdRPM(i)<=40 && absdRPM(i)>30
        RPMmagnitudechange(i)=5;
    elseif absdRPM(i)<=30 && absdRPM(i)>20
        RPMmagnitudechange(i)=4;
    elseif absdRPM(i)<=20 && absdRPM(i)>10
        RPMmagnitudechange(i)=3;
    elseif absdRPM(i)<=10 && absdRPM(i)>5
        RPMmagnitudechange(i)=2;
    elseif absdRPM(i)<=5
        RPMmagnitudechange(i)=1;
    end
```

```
end

RPMindicator=zeros(n,1);
for i=1:n %converts dRPM to categories (pos, neg, zero)
    if dRPM(i)<0
        RPMindicator(i)=-1;
    elseif dRPM(i)>0
        RPMindicator(i)=1;
    else
        RPMindicator(i)=0;
    end
end

EraticRPMIndicator=zeros(n,1);
for i=1:n-1
    if RPMindicator(i)-RPMindicator(i+1)==2 || RPMindicator(i)-
RPMindicator(i+1)==-2
        EraticRPMIndicator(i+1)=2; %oscillating between two consecutive points
    elseif RPMindicator(i)-RPMindicator(i+1)==0
        EraticRPMIndicator(i+1)=1; %continuing to increase or decrease between
two consecutive points
    elseif RPMindicator(i)-RPMindicator(i+1)==1 || RPMindicator(i)-
RPMindicator(i+1)==-1
            EraticRPMIndicator(i+1)=0; %stabilizing or plateauing between two
consecutive points
    end
end

MicroEraticRPM=zeros(n,1);
for i=1+span:n
    microinterval3= EraticRPMIndicator(i-span:i,1); %last 50 data points prior
to current of dTorque
    m=numel(microinterval3);
    osc=find(microinterval3==2);
    cont=find(microinterval3==1);
    stab=find(microinterval3==0);
    O=numel(osc);
    C=numel(cont);
    S=numel(stab);
    Oratio=O/m;
    if Oratio==0
        MicroEraticRPM(i)=0; %stable
    elseif Oratio>0 && Oratio<=.25
        MicroEraticRPM(i)=1; %fairly stable
    elseif Oratio>.25 && Oratio<=.5
        MicroEraticRPM(i)=2; %somewhat oscillating
```

```
    elseif Oratio>.5 && Oratio<=.75
        MicroEraticRPM(i)=3; %moderately oscillating
    elseif Oratio>.75 && Oratio<1
        MicroEraticRPM(i)=4; %highly oscillating
    elseif Oratio==1
        MicroEraticRPM(i)=5; %fully oscillating
    end
end


MSEdataclean=[MSEdataclean MicroEraticRPM];

clear dRPM microinterval3 i m osc cont stab O C S Oratio MicroEraticRPM
EraticRPMIndicator RPMindicator

%%Define erratic ROP behavior
dROP=MSEdataclean(:,40);
absdROP=abs(dROP);
ROPmagnitudechange=zeros(n,1);
for i=1:n
    if absdROP(i)>200
        ROPmagnitudechange(i)=7;
    elseif absdROP(i)<=200 && absdROP(i)>150
        ROPmagnitudechange(i)=6;
    elseif absdROP(i)<=150 && absdROP(i)>100
        ROPmagnitudechange(i)=5;
    elseif absdROP(i)<=100 && absdROP(i)>50
        ROPmagnitudechange(i)=4;
    elseif absdROP(i)<=50 && absdROP(i)>25
        ROPmagnitudechange(i)=3;
    elseif absdROP(i)<=25 && absdROP(i)>10
        ROPmagnitudechange(i)=2;
    elseif absdROP(i)<=10
        ROPmagnitudechange(i)=1;
    end
end

ROPindicator=zeros(n,1);
for i=1:n %converts dRPM to categories (pos, neg, zero)
    if dROP(i)<0
        ROPindicator(i)=-1;
    elseif dROP(i)>0
        ROPindicator(i)=1;
    else
        ROPindicator(i)=0;
    end
```

```
end

EraticROPIndicator=zeros(n,1);
for i=1:n-1
    if ROPindicator(i)-ROPindicator(i+1)==2 || ROPindicator(i)-
ROPindicator(i+1)==-2
        EraticROPIndicator(i+1)=2; %oscillating between two consecutive points
    elseif ROPindicator(i)-ROPindicator(i+1)==0
        EraticROPIndicator(i+1)=1; %continuing to increase or decrease between
two consecutive points
    elseif ROPindicator(i)-ROPindicator(i+1)==1 || ROPindicator(i)-
ROPindicator(i+1)==-1
        EraticROPIndicator(i+1)=0; %stabilizing or plateauing between two
consecutive points
    end
end

MicroEraticROP=zeros(n,1);
for i=1+span:n
    microinterval3= EraticROPIndicator(i-span:i,1); %last 50 data points prior
to current of dTorque
    m=numel(microinterval3);
    osc=find(microinterval3==2);
    cont=find(microinterval3==1);
    stab=find(microinterval3==0);
    O=numel(osc);
    C=numel(cont);
    S=numel(stab);
    Oratio=O/m;
    if Oratio==0
        MicroEraticROP(i)=0; %stable
    elseif Oratio>0 && Oratio<=.25
        MicroEraticROP(i)=1; %fairly stable
    elseif Oratio>.25 && Oratio<=.5
        MicroEraticROP(i)=2; %somewhat oscillating
    elseif Oratio>.5 && Oratio<=.75
        MicroEraticROP(i)=3; %moderately oscillating
    elseif Oratio>.75 && Oratio<1
        MicroEraticROP(i)=4; %highly oscillating
    elseif Oratio==1
        MicroEraticROP(i)=5; %fully oscillating
    end
end


MSEdataclean=[MSEdataclean MicroEraticROP];
```

```
%Add magnitude change data
MSEdataclean=[MSEdataclean Torquemagnitudechange WOBmagnitudechange
RPMmagnitudechange ROPmagnitudechange];

clear dROP microinterval3 i m osc cont stab O C S Oratio MicroEraticROP
EraticROPIndicator ROPindicator

clear absdTorque absdWOB absdRPM absdROP Torquemagnitudechange
WOBmagnitudechange RPMmagnitudechange ROPmagnitudechange n span dsmoothWOB
dsmoothROP dsmoothRPM dsmoothTorque RockStrengthChange EraticTorqueIndicator
MicroEraticTorque Torqueindicator Zeroratio
```

K. Additional Calculated Drilling Metrics

The program presented below is referred to in the master program in Appendix A as “MoreMetrics.m”.

```
%More calculated metrics for analysis
%%Coefficient of Friction using dROP and dWOB
dROP=MSEdataclean(:,40);
dWOB=MSEdataclean(:,43);
CF=dROP./dWOB;

MSEdataclean=[MSEdataclean CF];

%%Other ratios of derivatives
dRPM=MSEdataclean(:,41);
dTorque=MSEdataclean(:,42);

dROPvdRPM=dROP./dRPM;
dROPvdT=dROP./dTorque;

%%Depth of Cut
ROP=MSEdataclean(:,5);
RPM=MSEdataclean(:,26);
DOC=ROP./RPM;

MSEdataclean=[MSEdataclean dROPvdRPM dROPvdT DOC];

clear dROP dWOB CF dRPM dTorque dROPvdRPM dROPvdT ROP RPM DOC
```

L. Process Well 8 Curve for Automated Parameter Behavior Analysis

The program presented below imports the processed Well 8 master data file and further processes it for automated parameter behavior analysis, including for use in the Random Forest algorithm presented in the Appendices below.

```
%Pad 8 Curve processing for automated parameter behavior analysis

%%Trims to just curve and remap UCS to TVD
run('TrimTVDPad8Curve.m')

%%Creates vector of my hand classification of the curve section (not
%%necessary for analysis)
run('HandClassification.m')
%%Classify dUCSmapped based on + - or constant slope
run('dUCSclassification.m')

%%Create identity matrix with layers of parameter combinations and predict
%%UCS
run('createidentitymatrix.m')
run('LikelyUCS.m')
Pad8Curve=[Pad8Curve PredictedUCS PredictedUCS1stlayer PredictedUCS2ndlayer
Groups1 Groups2 SlopeMatrix];

run('PlotPredictedUCS.m')
Pad8Curve=[Pad8Curve UCS];


Models=[ROP ROP1 d T T1];
Pad8Curve=[Pad8Curve Models];

run('DataLabels.m')
Pad8Curve=[Pad8Curve Labelsadd ModeledMSECurve];

Pad8CurveTable=array2table(Pad8Curve);
Pad8CurveTable.Properties.VariableNames={'HoleDepth' 'WeightonBit' 'BitRPM'
'DifferentialPressure' 'RateofPenetration' 'MotorRPM' 'BlockHeight' 'RotaryRPM'
'TotalPumpOutput' 'BitDepth' 'RotaryTorque' 'Inclination' 'Azimuth'
'DysfunctionDepth' 'BitBallingBelief' 'BitBounceBelief' 'StickSlipBelief'
'WhirlBelief' 'OtherDysfunctionBelief' 'NoDysfunctionBelief' 'UCSanalog' 'BHA'
'MSE' 'MSEwithRotaryTorque' 'Time' 'CalcedBitRPM' 'DrillingEfficiency'
'BitAggressiveness' 'HSI' 'Torque' 'TVDSurvey' 'CourseNorthSurvey'
```

```
'CourseEastSurvey' 'TVD' 'CourseNorth' 'CourseEast' 'BinnedMSE' 'SmoothMSE'
'dROP' 'dRPM' 'dTorque' 'dWOB' 'dTime' 'dsmoothROP' 'dsmoothRPM'
'dsmoothTorque' 'dsmoothWOB' 'dsmoothTime' 'sWOBandTvsROP' 'ZeroRatio'
'MicroErraticTorque' 'MicroErraticWOB' 'MicroErraticRPM' 'MicroErraticROP'
'TorqueMagnitudeChange' 'WOBMagnitudeChange' 'RPMMagnitudeChange'
'ROPMagnitudeChange' 'sWOBandTandROP' 'sWOBandROPvsT' 'sTandROPvsWOB'
'WOBandTandROP' 'WOBandROPvsT' 'TandROPvsWOB' 'WOBandTvsROP' 'ROPandWOB'
'CoefficientofFriction' 'dROPvdRPM' 'dROPvdT' 'DOC' 'smoothTorque' 'smoothROP'
'smoothWOB' 'smoothRPM' 'smoothTime' 'Pad8UCS' 'UCSmapped' 'dUCSmapped' 'B'
'WOBestslope' 'ROPestslope' 'Testslope' 'NormalBehavior' 'UCSindicator'
'BinnedSmoothBehavior' 'BinnedSmoothUCSIndicator' 'WOBavgslope' 'ROPavgslope'
'Tavgslope' 'SmoothUCSindicator' 'MSEUCSdif' 'UCSvsMSE' 'MSEvsUCS' 'dMSE'
'dsmoothMSE' 'smoothUCSmapped' 'HandClassification' 'UCSMappedSlopeClass'
'LikelyUCS' 'Layer1' 'Layer2' 'Layer1Groups' 'Layer2Groups' 'WOBsloperaw'
'ROPsloperaw' 'Tsloperaw' 'WOBslopeclassraw' 'ROPslopeclassraw'
'Tslopeclassraw' 'WOBslopeSM' 'ROPslopeSM' 'TslopeSM' 'WOBslopeclassSM'
'ROPslopeclassSM' 'TslopeclassSM' 'PlottedPredictedUCS' 'TimeSeconds'
'timeindicator' 'instantWOBslope' 'instantROPslope' 'instantTslope'
'instWOBclass' 'instROPclass' 'instTclass' 'UCSslopeclassification' 'WOBchar'
'ROPchar' 'Tchar'};
writetable(Pad8CurveTable,'C:\Users\Carolyn\Documents\Masters Research\Spring
2018\Apache\Processed Single Files\Pad8Curve.csv')
```

M. TRIM WELL 8 DATA SET TO CURVE SECTION COVERED BY UCS DATA

The program presented below is referred to in the Well 8 processing program in Appendix L as “TrimTVDPad8Curve.m”.

```
%Trim the Pad 8 data to only include the curve section over which UCS data
%was taken
%%Re-Import UCS
opts=detectImportOptions('UCS.xlsx');
opts.SelectedVariableNames={'DEPTH' 'UCS_i'};
A=readtable('UCS.xlsx',opts);
UCS8=table2array(A);
clear A

%%Find TVD where UCS started being taken and where it ended
TVDindex=find(UCS8(:,2)>0);
UCSTVD=UCS8(TVDindex,1);
```

```
TVDstart=UCSTVD(1);
TVDend=UCSTVD(end);

%%Trim MSE matrix to be just within BHA #23 (curve BHA)
BHAindex=find(Pad8(:,22)==23);
Pad8Curve=Pad8(BHAindex,:);

%%Trim to be just MSE post time gap (curve drilling)
Timeindex=find(Pad8Curve(:,25)>800);
Pad8Curve=Pad8Curve(Timeindex,:);

%%Normalize TVD values to .5 ft scale to match UCS holedepth scale
m=numel(Pad8Curve(:,35));
TVD=Pad8Curve(:,35);
rounded=round(TVD);
dif=TVD-rounded;
adjusteddepth=rounded;
for i=1:m
    if dif(i)>0 && dif(i)<.25
        adjusteddepth(i)=rounded(i);
    elseif dif(i)>0 && dif(i)>=.25
            adjusteddepth(i)=rounded(i)+.5;
    elseif dif(i)<0 && dif(i)<-.25
                adjusteddepth(i)=rounded(i)-.5;
    elseif dif(i)<0 && dif(i)>=-.25
                    adjusteddepth(i)=rounded(i);
   end
end

%%Map UCS to MSE matrix TVD
m=numel(Pad8Curve(:,1));
UCS8mapped=zeros(m,1);
for i=1:m
    A=adjusteddepth(i);
    UCS8index=find(UCS8(:,1)==A);
    B=UCS8(UCS8index,2);
    UCS8mapped(i)=B;
end

%%Map UCS differential to TVD
dUCSmapped=zeros(m,1);
dUCS=diff(UCS8(:,2));
dUCS=[0;dUCS];
for i=1:m
    A=adjusteddepth(i);
    dUCSindex=find(UCS8(:,1)==A);
```

```
    B=dUCS(dUCSindex);
    dUCSmapped(i)=B;
end
Pad8Curve=[Pad8Curve UCS8mapped dUCSmapped];

%%Trim to just where UCS data was taken
UCSstart=find(Pad8Curve(:,78)>0); %must update this everytime I add a new
metric or variable to full well matrix (+2 from final row)
Pad8Curve=Pad8Curve(UCSstart,:);

%%Run Parameter Behavior binning code
run('ParameterBehavior.m')

%%More UCS and MSE metrics
MSE=Pad8Curve(:,23);
UCS=Pad8Curve(:,78); %must update this everytime I add a new metric or variable
to full well matrix (+2 from final row)
MSEUCSdif=MSE-UCS;
UCSvsMSE=UCS./MSE;
MSEvsUCS=MSE./UCS;

Pad8Curve=[Pad8Curve MSEUCSdif UCSvsMSE MSEvsUCS];
clear dUCSindex dUCSmapped dUCS UCSstart MSEvsUCS MSE UCS MSEUCSdif UCSvsMSE A
adjusteddepth B dif i m opts rounded Timeindex TVD TVDend TVDindex TVDstart
UCS8 UCS8index UCS8mapped UCSTVD X Y BHAindex

dMSE=diff(Pad8Curve(:,23));
dMSE=[0; dMSE];
dsmoothMSE=diff(Pad8Curve(:,39));
dsmoothMSE=[0; dsmoothMSE];
smoothUCSmapped=smooth(Pad8Curve(:,78),20);
Pad8Curve=[Pad8Curve dMSE dsmoothMSE smoothUCSmapped];
clear dMSE dsmoothMSE smoothUCSmapped
```

N. Parameter Behavior Classification per Initial Hypotheses

The program presented below is referred to in the Well 8 trimming program presented in Appendix M as “ParameterBehavior.m”.

```
%Parameter Behavior Definition and Hand Classification, per initial Hypotheses

%%Split MSE output into time based sections and take average over each section
%%Create bin assignments for time data
R=5;
```

```
W=1;
T=300;
bin=input('Input Time scale to bin [min] = ');
bin=bin*60; %puts bin size into seconds
binpoints=bin/10; %gives # of time data points in bin
if binpoints < 3 %30 secs (3 data points) is lowest scale to operate average
slope on, smaller should just use smoothed data
       bin=input('Time scale too small, input new time scale [min] = ');
       bin=bin*60;
       binpoints=bin/10;
end
m=numel(Pad8Curve(:,25)); %number of time data points total
bins=m/binpoints;%number of bins to break up time data into
bins=ceil(bins); %rounds the # of bins to next highest integer to use in
discretize function
B=repmat(1:bins,binpoints,1);
B=B(1:end)'; %creates bin indeces vector
n=numel(B);
dif=n-m;
B=B(1:end-dif); %resized bin vector to be same size at Pad8Curve

%%Calculate average slope using average of first and last few points, if range
is larger than 9 use avg of first and last 5 data points for slope, if equal to
or less than 9 take avg of first and last 2 points of each and take difference
WOBslope=zeros(m,1);
ROPslope=zeros(m,1);
Tslope=zeros(m,1);
behavior=zeros(m,1);
UCSindicator=zeros(m,1);
Bsmoothbehavior=zeros(m,1);
UCSindicatorsmoothB=zeros(m,1);
if binpoints>=10
    for j=1:bins-1
        %Bin using raw data
        index=find(B(:,1)==j);
        binWOB=Pad8Curve(index,2);
        firstWOB=binWOB(1:5); %takes first and last 5 points to average
        avgfirstWOB=mean(firstWOB);
        lastWOB=binWOB(end-4:end);
        avglastWOB=mean(lastWOB);
        slopeWOB=avglastWOB-avgfirstWOB;
        if slopeWOB>-W && slopeWOB<W %Redefine low slopes as zero slope
(constant)
            slopeWOB=0;
        end
        WOBslope(index,1)=slopeWOB;
```

```
        binROP=Pad8Curve(index,5);
        firstROP=binROP(1:5);
        avgfirstROP=mean(firstROP);
        lastROP=binROP(end-4:end);
        avglastROP=mean(lastROP);
        slopeROP=avglastROP-avgfirstROP;
        if slopeROP>-R && slopeROP<R %Redefine low slopes as zero slope
(constant)
            slopeROP=0;
        end
        ROPslope(index,1)=slopeROP;

        binT=Pad8Curve(index,31);
        firstT=binT(1:5);
        avgfirstT=mean(firstT);
        lastT=binT(end-4:end);
        avglastT=mean(lastT);
        slopeT=avglastT-avgfirstT;
        if slopeT>-T && slopeT<T %Redefine low slopes as zero slope (constant)
            slopeT=0;
        end
        Tslope(index,1)=slopeT;

       %Classify slope combinations as "Normal"=1 or "Abnormal"=0
        if slopeWOB>0 && slopeT>0 && slopeROP>0
            behavior(index,1)=1; %1=Normal
        elseif slopeWOB==0 && slopeT==0 && slopeROP==0
            behavior(index,1)=1; %also normal
        elseif slopeWOB<0 && slopeT<0 && slopeROP<0
            behavior(index,1)=1; %also normal
        end

        %Combo UCS and WOB effects for normal behavior
        if slopeWOB>0 && slopeROP>0 && slopeT>0
            UCSindicator(index,1)=0; %0 means constant UCS
        elseif slopeWOB<0 && slopeROP<0 && slopeT<0
            UCSindicator(index,1)=0; %also constant
        elseif slopeWOB>0 && slopeROP==0 && slopeT>0
            UCSindicator(index,1)=1; %UCS increasing
        elseif slopeWOB<0 && slopeROP==0 && slopeT<0
            UCSindicator(index,1)=-1; %UCS decreasing
        elseif slopeWOB<0 && slopeROP<0 && slopeT==0
            UCSindicator(index,1)=1; %UCS increasing
        elseif slopeWOB>0 && slopeROP>0 && slopeT==0
            UCSindicator(index,1)=-1; %UCS decreasing
```

```
        elseif slopeWOB==0 && slopeROP==0 && slopeT==00
            UCSindicator(index,1)=0; %UCS constant
        elseif slopeWOB==0 && slopeROP<0 && slopeT>0
            UCSindicator(index,1)=1; %UCS increasing
        elseif slopeWOB==0 && slopeROP>0 && slopeT<0
            UCSindicator(index,1)=-1; %UCS decreasing
        else
            UCSindicator(index,1)=2; %potentially abnormal
        end

        %%Bin using smooth data
        index=find(B(:,1)==j);
        binWOB=Pad8Curve(index,74);
        firstWOB=binWOB(1:5); %takes first and last 5 points to average
        avgfirstWOB=mean(firstWOB);
        lastWOB=binWOB(end-4:end);
        avglastWOB=mean(lastWOB);
        slopeWOB=avglastWOB-avgfirstWOB;
        if slopeWOB>-W && slopeWOB<W %Redefine low slopes as zero slope
(constant)
            slopeWOB=0;
        end
        WOBslope(index,1)=slopeWOB;

        binROP=Pad8Curve(index,73);
        firstROP=binROP(1:5);
        avgfirstROP=mean(firstROP);
        lastROP=binROP(end-4:end);
        avglastROP=mean(lastROP);
        slopeROP=avglastROP-avgfirstROP;
        if slopeROP>-R && slopeROP<R %Redefine low slopes as zero slope
(constant)
            slopeROP=0;
        end
        ROPslope(index,1)=slopeROP;

        binT=Pad8Curve(index,72);
        firstT=binT(1:5);
        avgfirstT=mean(firstT);
        lastT=binT(end-4:end);
        avglastT=mean(lastT);
        slopeT=avglastT-avgfirstT;
        if slopeT>-T && slopeT<T %Redefine low slopes as zero slope (constant)
            slopeT=0;
        end
        Tslope(index,1)=slopeT;
```

```
       %Classify slope combinations as "Normal"=1 or "Abnormal"=0
        if slopeWOB>0 && slopeT>0 && slopeROP>0
            Bsmoothbehavior(index,1)=1; %1=Normal
        elseif slopeWOB==0 && slopeT==0 && slopeROP==0
            Bsmoothbehavior(index,1)=1; %also normal
        elseif slopeWOB<0 && slopeT<0 && slopeROP<0
            Bsmoothbehavior(index,1)=1; %also normal
        end

        %Combo UCS and WOB effects for normal behavior
        if slopeWOB>0 && slopeROP>0 && slopeT>0
            UCSindicatorsmoothB(index,1)=0; %0 means constant UCS
        elseif slopeWOB<0 && slopeROP<0 && slopeT<0
            UCSindicatorsmoothB(index,1)=0; %also constant
        elseif slopeWOB>0 && slopeROP==0 && slopeT>0
            UCSindicatorsmoothB(index,1)=1; %UCS increasing
        elseif slopeWOB<0 && slopeROP==0 && slopeT<0
            UCSindicatorsmoothB(index,1)=-1; %UCS decreasing
        elseif slopeWOB<0 && slopeROP<0 && slopeT==0
            UCSindicatorsmoothB(index,1)=1; %UCS increasing
        elseif slopeWOB>0 && slopeROP>0 && slopeT==0
            UCSindicatorsmoothB(index,1)=-1; %UCS decreasing
        elseif slopeWOB==0 && slopeROP==0 && slopeT==00
            UCSindicatorsmoothB(index,1)=0; %UCS constant
        elseif slopeWOB==0 && slopeROP<0 && slopeT>0
            UCSindicatorsmoothB(index,1)=1; %UCS increasing
        elseif slopeWOB==0 && slopeROP>0 && slopeT<0
            UCSindicatorsmoothB(index,1)=-1; %UCS decreasing
        else
            UCSindicatorsmoothB(index,1)=2; %potentially abnormal
        end

    end
end
if binpoints<10 && binpoints>=4
    for j=1:bins-1
        %%Bin using raw data
        index=find(B(:,1)==j);
        binWOB=Pad8Curve(index,2);
        firstWOB=binWOB(1:2); %takes first and last 2 points to average and get
slope
        avgfirstWOB=mean(firstWOB);
        lastWOB=binWOB(end-1:end);
        avglastWOB=mean(lastWOB);
        slopeWOB=avglastWOB-avgfirstWOB;
```

```
        if slopeWOB>-W && slopeWOB<W %Redefine low slopes as zero slope
(constant)
            slopeWOB=0;
        end
        WOBslope(index,1)=slopeWOB;

        binROP=Pad8Curve(index,5);
        firstROP=binROP(1:2);
        avgfirstROP=mean(firstROP);
        lastROP=binROP(end-1:end);
        avglastROP=mean(lastROP);
        slopeROP=avglastROP-avgfirstROP;
        if slopeROP>-R && slopeROP<R %Redefine low slopes as zero slope
(constant)
            slopeROP=0;
        end
        ROPslope(index,1)=slopeROP;

        binT=Pad8Curve(index,31);
        firstT=binT(1:2);
        avgfirstT=mean(firstT);
        lastT=binT(end-1:end);
        avglastT=mean(lastT);
        slopeT=avglastT-avgfirstT;
        if slopeT>-T && slopeT<T %Redefine low slopes as zero slope (constant)
            slopeT=0;
        end
        Tslope(index,1)=slopeT;

        %Classify slope combinations as "Normal"=1 or "Abnormal"=0
        if slopeWOB>0 && slopeT>0 && slopeROP>0
            behavior(index,1)=1; %1=Normal
        elseif slopeWOB==0 && slopeT==0 && slopeROP==0
            behavior(index,1)=1; %also normal
        elseif slopeWOB<0 && slopeT<0 && slopeROP<0
            behavior(index,1)=1; %also normal
        end

           %Combo UCS and WOB effects for normal behavior
        if slopeWOB>0 && slopeROP>0 && slopeT>0
            UCSindicator(index,1)=0; %0 means constant UCS
        elseif slopeWOB<0 && slopeROP<0 && slopeT<0
            UCSindicator(index,1)=0; %also constant
        elseif slopeWOB>0 && slopeROP==0 && slopeT>0
            UCSindicator(index,1)=1; %UCS increasing
        elseif slopeWOB<0 && slopeROP==0 && slopeT<0
```

```
            UCSindicator(index,1)=-1; %UCS decreasing
        elseif slopeWOB<0 && slopeROP<0 && slopeT==0
            UCSindicator(index,1)=1; %UCS increasing
        elseif slopeWOB>0 && slopeROP>0 && slopeT==0
            UCSindicator(index,1)=-1; %UCS decreasing
        elseif slopeWOB==0 && slopeROP==0 && slopeT==00
            UCSindicator(index,1)=0; %UCS constant
        elseif slopeWOB==0 && slopeROP<0 && slopeT>0
            UCSindicator(index,1)=1; %UCS increasing
        elseif slopeWOB==0 && slopeROP>0 && slopeT<0
            UCSindicator(index,1)=-1; %UCS decreasing
        else
            UCSindicator(index,1)=2; %potentially abnormal
        end


        %%Bin using smooth data
        index=find(B(:,1)==j);
        binWOB=Pad8Curve(index,74);
        firstWOB=binWOB(1:2); %takes first and last 2 points to average and get
slope
        avgfirstWOB=mean(firstWOB);
        lastWOB=binWOB(end-1:end);
        avglastWOB=mean(lastWOB);
        slopeWOB=avglastWOB-avgfirstWOB;
        if slopeWOB>-W && slopeWOB<W %Redefine low slopes as zero slope
(constant)
            slopeWOB=0;
        end
        WOBslope(index,1)=slopeWOB;

        binROP=Pad8Curve(index,73);
        firstROP=binROP(1:2);
        avgfirstROP=mean(firstROP);
        lastROP=binROP(end-1:end);
        avglastROP=mean(lastROP);
        slopeROP=avglastROP-avgfirstROP;
        if slopeROP>-R && slopeROP<R %Redefine low slopes as zero slope
(constant)
            slopeROP=0;
        end
        ROPslope(index,1)=slopeROP;

        binT=Pad8Curve(index,72);
        firstT=binT(1:2);
        avgfirstT=mean(firstT);
```

```
        lastT=binT(end-1:end);
        avglastT=mean(lastT);
        slopeT=avglastT-avgfirstT;
        if slopeT>-T && slopeT<T %Redefine low slopes as zero slope (constant)
            slopeT=0;
        end
        Tslope(index,1)=slopeT;

        %Classify slope combinations as "Normal"=1 or "Abnormal"=0
        if slopeWOB>0 && slopeT>0 && slopeROP>0
            Bsmoothbehavior(index,1)=1; %1=Normal
        elseif slopeWOB==0 && slopeT==0 && slopeROP==0
            Bsmoothbehavior(index,1)=1; %also normal
        elseif slopeWOB<0 && slopeT<0 && slopeROP<0
            Bsmoothbehavior(index,1)=1; %also normal
        end

        %Combo UCS and WOB effects for normal behavior
        if slopeWOB>0 && slopeROP>0 && slopeT>0
            UCSindicatorsmoothB(index,1)=0; %0 means constant UCS
        elseif slopeWOB<0 && slopeROP<0 && slopeT<0
            UCSindicatorsmoothB(index,1)=0; %also constant
        elseif slopeWOB>0 && slopeROP==0 && slopeT>0
            UCSindicatorsmoothB(index,1)=1; %UCS increasing
        elseif slopeWOB<0 && slopeROP==0 && slopeT<0
            UCSindicatorsmoothB(index,1)=-1; %UCS decreasing
        elseif slopeWOB<0 && slopeROP<0 && slopeT==0
            UCSindicatorsmoothB(index,1)=1; %UCS increasing
        elseif slopeWOB>0 && slopeROP>0 && slopeT==0
            UCSindicatorsmoothB(index,1)=-1; %UCS decreasing
        elseif slopeWOB==0 && slopeROP==0 && slopeT==00
            UCSindicatorsmoothB(index,1)=0; %UCS constant
        elseif slopeWOB==0 && slopeROP<0 && slopeT>0
            UCSindicatorsmoothB(index,1)=1; %UCS increasing
        elseif slopeWOB==0 && slopeROP>0 && slopeT<0
            UCSindicatorsmoothB(index,1)=-1; %UCS decreasing
        else
            UCSindicatorsmoothB(index,1)=2; %potentially abnormal
        end

    end
end
if binpoints<=3
    for j=1:bins-1
        %%Bin using raw data
        index=find(B(:,1)==j);
```

```
        binWOB=Pad8Curve(index,2);
        firstWOB=binWOB(1); %takes first point and last point and then takes
slope betwee
        lastWOB=binWOB(end);
        slopeWOB=lastWOB-firstWOB;
        if slopeWOB>-W && slopeWOB<W %Redefine low slopes as zero slope
(constant)
            slopeWOB=0;
        end
        WOBslope(index,1)=slopeWOB;

        binROP=Pad8Curve(index,5);
        firstROP=binROP(1);
        lastROP=binROP(end);
        slopeROP=lastROP-firstROP;
        if slopeROP>-R && slopeROP<R %Redefine low slopes as zero slope
(constant)
            slopeROP=0;
        end
        ROPslope(index,1)=slopeROP;

        binT=Pad8Curve(index,31);
        firstT=binT(1);
        lastT=binT(end);
        slopeT=lastT-firstT;
        if slopeT>-T && slopeT<T %Redefine low slopes as zero slope (constant)
            slopeT=0;
        end
        Tslope(index,1)=slopeT;

        %Classify slope combinations as "Normal"=1 or "Abnormal"=0
        if slopeWOB>0 && slopeT>0 && slopeROP>0
            behavior(index,1)=1; %1=Normal
        elseif slopeWOB==0 && slopeT==0 && slopeROP==0
            behavior(index,1)=1; %also normal
        elseif slopeWOB<0 && slopeT<0 && slopeROP<0
            behavior(index,1)=1; %also normal
        end

            %Combo UCS and WOB effects for normal behavior
        if slopeWOB>0 && slopeROP>0 && slopeT>0
            UCSindicator(index,1)=0; %0 means constant UCS
        elseif slopeWOB<0 && slopeROP<0 && slopeT<0
            UCSindicator(index,1)=0; %also constant
        elseif slopeWOB>0 && slopeROP==0 && slopeT>0
            UCSindicator(index,1)=1; %UCS increasing
```

```
        elseif slopeWOB<0 && slopeROP==0 && slopeT<0
            UCSindicator(index,1)=-1; %UCS decreasing
        elseif slopeWOB<0 && slopeROP<0 && slopeT==0
            UCSindicator(index,1)=1; %UCS increasing
        elseif slopeWOB>0 && slopeROP>0 && slopeT==0
            UCSindicator(index,1)=-1; %UCS decreasing
        elseif slopeWOB==0 && slopeROP==0 && slopeT==00
            UCSindicator(index,1)=0; %UCS constant
        elseif slopeWOB==0 && slopeROP<0 && slopeT>0
            UCSindicator(index,1)=1; %UCS increasing
        elseif slopeWOB==0 && slopeROP>0 && slopeT<0
            UCSindicator(index,1)=-1; %UCS decreasing
        else
            UCSindicator(index,1)=2; %potentially abnormal
        end

        %%Bin using smooth data
        index=find(B(:,1)==j);
        binWOB=Pad8Curve(index,2);
        firstWOB=binWOB(1); %takes first point and last point and then takes
slope betwee
        lastWOB=binWOB(end);
        slopeWOB=lastWOB-firstWOB;
        if slopeWOB>-W && slopeWOB<W %Redefine low slopes as zero slope
(constant)
            slopeWOB=0;
        end
        WOBslope(index,1)=slopeWOB;

        binROP=Pad8Curve(index,5);
        firstROP=binROP(1);
        lastROP=binROP(end);
        slopeROP=lastROP-firstROP;
        if slopeROP>-R && slopeROP<R %Redefine low slopes as zero slope
(constant)
            slopeROP=0;
        end
        ROPslope(index,1)=slopeROP;

        binT=Pad8Curve(index,31);
        firstT=binT(1);
        lastT=binT(end);
        slopeT=lastT-firstT;
        if slopeT>-T && slopeT<T %Redefine low slopes as zero slope (constant)
            slopeT=0;
        end
```

```
        Tslope(index,1)=slopeT;


        %Classify slope combinations as "Normal"=1 or "Abnormal"=0
        if slopeWOB>0 && slopeT>0 && slopeROP>0
            Bsmoothbehavior(index,1)=1; %1=Normal
        elseif slopeWOB==0 && slopeT==0 && slopeROP==0
            Bsmoothbehavior(index,1)=1; %also normal
        elseif slopeWOB<0 && slopeT<0 && slopeROP<0
            Bsmoothbehavior(index,1)=1; %also normal
        end

        %Combo UCS and WOB effects for normal behavior
        if slopeWOB>0 && slopeROP>0 && slopeT>0
            UCSindicatorsmoothB(index,1)=0; %0 means constant UCS
        elseif slopeWOB<0 && slopeROP<0 && slopeT<0
            UCSindicatorsmoothB(index,1)=0; %also constant
        elseif slopeWOB>0 && slopeROP==0 && slopeT>0
            UCSindicatorsmoothB(index,1)=1; %UCS increasing
        elseif slopeWOB<0 && slopeROP==0 && slopeT<0
            UCSindicatorsmoothB(index,1)=-1; %UCS decreasing
        elseif slopeWOB<0 && slopeROP<0 && slopeT==0
            UCSindicatorsmoothB(index,1)=1; %UCS increasing
        elseif slopeWOB>0 && slopeROP>0 && slopeT==0
            UCSindicatorsmoothB(index,1)=-1; %UCS decreasing
        elseif slopeWOB==0 && slopeROP==0 && slopeT==00
            UCSindicatorsmoothB(index,1)=0; %UCS constant
        elseif slopeWOB==0 && slopeROP<0 && slopeT>0
            UCSindicatorsmoothB(index,1)=1; %UCS increasing
        elseif slopeWOB==0 && slopeROP>0 && slopeT<0
            UCSindicatorsmoothB(index,1)=-1; %UCS decreasing
        else
            UCSindicatorsmoothB(index,1)=2; %potentially abnormal
        end

    end
end


Pad8Curve=[Pad8Curve B WOBslope ROPslope Tslope behavior UCSindicator
Bsmoothbehavior UCSindicatorsmoothB];

clear UCSindicatorsmoothB Bsmoothbehavior UCSindicator m j behavior slopeWOB
WOBslope ROPslope slopeROP Tslope slopeT index firstT lastT firstROP lastROP
firstWOB lastWOB binWOB binROP binT binpoints bin bins avgfirstROP avglastROP
avgfirstT avglastT avgfirstWOB avglastWOB
```

```
%%Can also take the difference between each data point (slope between each
%%point) and then bin and take an average (get average slope in bin)

WOB=Pad8Curve(:,2);
difW=diff(WOB);
difW=[0;difW];
binWslope=splitapply(@mean,difW,B);  %using bins from beginning of code
m=numel(difW(:,1));
extendedbinWslope=ones(m,1);
n=numel(binWslope);
for i=1:n
    index=find(B(:,1)==n);
    extendedbinWslope(index,1)=binWslope(i,1);
end

ROP=Pad8Curve(:,5);
difROP=diff(ROP);
difROP=[0;difROP];
binROPslope=splitapply(@mean,difROP,B);  %using bins from beginning of code
m=numel(difROP(:,1));
extendedbinROPslope=ones(m,1);
n=numel(binROPslope);
for i=1:n
    index=find(B(:,1)==n);
    extendedbinROPslope(index,1)=binROPslope(i,1);
end

T=Pad8Curve(:,31);
difT=diff(T);
difT=[0;difT];
binTslope=splitapply(@mean,difT,B);  %using bins from beginning of code
m=numel(difT(:,1));
extendedbinTslope=ones(m,1);
n=numel(binTslope);
for i=1:n
    index=find(B(:,1)==n);
    extendedbinTslope(index,1)=binTslope(i,1);
end



Pad8Curve=[Pad8Curve extendedbinWslope extendedbinROPslope extendedbinTslope];
```

```
clear WOB ROP T m n i difW difROP difT index B binWslope binTslope binROPslope
extendedbinWslope extendedbinROPslope extendedbinTslope


%%Generate UCS indicator using point by point derviatives from the smoothed MSE
vector (typically smoothed over 2 min, but can vary and will change this
output)
dsmoothWOB=Pad8Curve(:,48);
dsmoothROP=Pad8Curve(:,45);
dsmoothT=Pad8Curve(:,47);
n=numel(Pad8Curve(:,48));
smoothUCSindicator=zeros(n,1);
%Redefine slopes as equivalent zero (constant) if within ranges
for i=1:n
    if dsmoothT(i)>-25 && dsmoothT(i)<25 %Redefine low slopes as zero slope
(constant)
        dsmoothT(i)=0;
    end
    if dsmoothROP(i)>-1 && dsmoothROP(i)<1 %Redefine low slopes as zero slope
(constant)
        dsmoothROP(i)=0;
    end
    if dsmoothWOB(i)>-.05 && dsmoothWOB(i)<.05 %Redefine low slopes as zero
slope (constant)
        dsmoothWOB(i)=0;
    end

        if dsmoothWOB(i)>0 && dsmoothROP(i)>0 && dsmoothT(i)>0
            smoothUCSindicator(i)=0; %0 means constant UCS
        elseif dsmoothWOB(i)<0 && dsmoothROP(i)<0 && dsmoothT(i)<0
            smoothUCSindicator(i)=0; %also constant
        elseif dsmoothWOB(i)>0 && dsmoothROP(i)==0 && dsmoothT(i)>0
            smoothUCSindicator(i)=1; %UCS increasing
        elseif dsmoothWOB(i)<0 && dsmoothROP(i)==0 && dsmoothT(i)<0
            smoothUCSindicator(i)=-1; %UCS decreasing
        elseif dsmoothWOB(i)<0 && dsmoothROP(i)<0 && dsmoothT(i)==0
            smoothUCSindicator(i)=1; %UCS increasing
        elseif dsmoothWOB(i)>0 && dsmoothROP(i)>0 && dsmoothT(i)==0
            smoothUCSindicator(i)=-1; %UCS decreasing
        elseif dsmoothWOB(i)==0 && dsmoothROP(i)==0 && dsmoothT(i)==00
            smoothUCSindicator(i)=0; %UCS constant
        elseif dsmoothWOB(i)==0 && dsmoothROP(i)<0 && dsmoothT(i)>0
            smoothUCSindicator(i)=1; %UCS increasing
        elseif dsmoothWOB(i)==0 && dsmoothROP(i)>0 && dsmoothT(i)<0
            smoothUCSindicator(i)=-1; %UCS decreasing
        else
```

```
            smoothUCSindicator(i)=2; %potentially abnormal
        end
end
Pad8Curve=[Pad8Curve smoothUCSindicator];

clear n smoothUCSindicator dsmoothWOB dsmoothROP dsmoothT dif R T W
```

O. Define UCS Derivative Classifications

The program presented below is referred to in the Well 8 curve processing program in Appendix L as “dUCSclassification.m”.

```
%Convert dUCS into classifications
n=numel(Pad8Curve(:,79));
ClassdUCSMapped=zeros(n,1);
for i=1:n
    if Pad8Curve(i,79)>50
        ClassdUCSMapped(i)=1; %if derivative of UCS is greater than 100 then
UCS is increasing (approximates constant UCS as -100<dUCS<100
    elseif Pad8Curve(i,79)<-50
        ClassdUCSMapped(i)=-1;
    else
        ClassdUCSMapped(i)=0;
    end
end

Pad8Curve=[Pad8Curve ClassdUCSMapped];

clear ClassdUCSMapped n i
```

P. Create CBM: Master Program

The program presented below is the master program for creating the CBM and is referred to in the Well 8 curve processing program as “createidentitymatrix.m”.

```
%Create CBM (aka Identity Matrix)
    %%Bin data
        bin=input('Input Time scale to bin [min] = ');
        bin=bin*60; %puts bin size into seconds
        binpoints=bin/10; %gives # of time data points in bin
        if binpoints < 3 %30 secs (3 data points) is lowest scale to operate
average slope on, smaller should just use smoothed data
```

```
                bin=input('Time scale too small, input new time scale [min] =
');
                bin=bin*60;
                binpoints=bin/10;
        end
        m=numel(Pad8Curve(:,25)); %number of time data points total
        bins=m/binpoints;%number of bins to break up time data into
        bins=ceil(bins); %rounds the # of bins to next highest integer to use
in discretize function
        B=repmat(1:bins,binpoints,1);
        B=B(1:end)'; %creates bin indeces vector
        n=numel(B);
        dif=n-m;
        B=B(1:end-dif); %resized bin vector to be same size at Pad8Curve
    %%First Layer
        %First column is WOB, ROP, and Torque slope identifiers
        run('slopeidentifiers.m') %COMPLETE, NEED TO TEST AND DEBUG
    %%Second Layer
        %Second and third columns are which parameters are constant (2nd is #
that are constant
        %and 3rd is which parameters)
        run('constantparam.m') %COMPLETED 2ND COLUMN, may not need 3rd
        %Fourth column is which line of associated slope magnitude combination
        %(based on column 2)
        run('mag.m')
        run('slopemagnitude.m')
    %%Third Layer
```

Q. First Column of the CBM: Parameter Slope Combinations

The program presented below is referred to in the master program for creating the CBM in Appendix P as “slopeidentifiers.m”.

```
%Create first column of identity matrix: identify slope combination
possibilities

%%Equivalent zero thresholds
    R=2;
    W=.6;
    T=175;
```

```
%%Calculate average slope using average of first and last few points, if range
is larger than 9 use avg of first and last 5 data points for slope, if equal to
or less than 9 take avg of first and last 2 points of each and take difference
WOBslope=zeros(m,1);
ROPslope=zeros(m,1);
Tslope=zeros(m,1);
sWOBslope=zeros(m,1);
sROPslope=zeros(m,1);
sTslope=zeros(m,1);
UCSindicator=zeros(m,1);
sUCSindicator=zeros(m,1);
WOBslopeclass=zeros(m,1);
WOBslopeclasssm=zeros(m,1);
ROPslopeclass=zeros(m,1);
ROPslopeclasssm=zeros(m,1);
Tslopeclass=zeros(m,1);
Tslopeclasssm=zeros(m,1);
if binpoints>=10
    for j=1:bins-1
        %Bin using raw data
        index=find(B(:,1)==j);
        binWOB=Pad8Curve(index,2);
        firstWOB=binWOB(1:5); %takes first and last 5 points to average
        avgfirstWOB=mean(firstWOB);
        lastWOB=binWOB(end-4:end);
        avglastWOB=mean(lastWOB);
        slopeWOB=avglastWOB-avgfirstWOB;
        if slopeWOB>-W && slopeWOB<W %Redefine low slopes as zero slope
(constant)
            slopeWOB=0;
        end
        WOBslope(index,1)=slopeWOB;
        if slopeWOB>0
            WOBslopeclass(index,1)=1;
        elseif slopeWOB<0
            WOBslopeclass(index,1)=-1;
        else
            WOBslopeclass(index,1)=0;
        end

        binROP=Pad8Curve(index,5);
        firstROP=binROP(1:5);
        avgfirstROP=mean(firstROP);
        lastROP=binROP(end-4:end);
        avglastROP=mean(lastROP);
        slopeROP=avglastROP-avgfirstROP;
```

```
        if slopeROP>-R && slopeROP<R %Redefine low slopes as zero slope
(constant)
            slopeROP=0;
        end
        ROPslope(index,1)=slopeROP;
        if slopeROP>0
            ROPslopeclass(index,1)=1;
        elseif slopeROP<0
            ROPslopeclass(index,1)=-1;
        else
            ROPslopeclass(index,1)=0;
        end

        binT=Pad8Curve(index,31);
        firstT=binT(1:5);
        avgfirstT=mean(firstT);
        lastT=binT(end-4:end);
        avglastT=mean(lastT);
        slopeT=avglastT-avgfirstT;
        if slopeT>-T && slopeT<T %Redefine low slopes as zero slope (constant)
            slopeT=0;
        end
        Tslope(index,1)=slopeT;
        if slopeT>0
            Tslopeclass(index,1)=1;
        elseif slopeT<0
            Tslopeclass(index,1)=-1;
        else
            Tslopeclass(index,1)=0;
        end

        %All possible parameter slope combinations
        if slopeWOB>0 && slopeROP>0 && slopeT>0
            UCSindicator(index,1)=1;
        elseif slopeWOB<0 && slopeROP<0 && slopeT<0
            UCSindicator(index,1)=2;
        elseif slopeWOB>0 && slopeROP==0 && slopeT>0
            UCSindicator(index,1)=3;
        elseif slopeWOB<0 && slopeROP==0 && slopeT<0
            UCSindicator(index,1)=4;
        elseif slopeWOB<0 && slopeROP<0 && slopeT==0
            UCSindicator(index,1)=5;
        elseif slopeWOB>0 && slopeROP>0 && slopeT==0
            UCSindicator(index,1)=6;
        elseif slopeWOB==0 && slopeROP==0 && slopeT==0
            UCSindicator(index,1)=7;
```

```
        elseif slopeWOB==0 && slopeROP<0 && slopeT>0
            UCSindicator(index,1)=8;
        elseif slopeWOB==0 && slopeROP>0 && slopeT<0
            UCSindicator(index,1)=9;
        elseif slopeWOB>0 && slopeROP<0 && slopeT<0
            UCSindicator(index,1)=10;
        elseif slopeWOB>0 && slopeROP==0 && slopeT==0
            UCSindicator(index,1)=11;
        elseif slopeWOB>0 && slopeROP<0 && slopeT==0
            UCSindicator(index,1)=12;
        elseif slopeWOB>0 && slopeROP==0 && slopeT<0
            UCSindicator(index,1)=13;
        elseif slopeWOB>0 && slopeROP<0 && slopeT>0
            UCSindicator(index,1)=14;
        elseif slopeWOB>0 && slopeROP>0 && slopeT<0
            UCSindicator(index,1)=15;
        elseif slopeWOB<0 && slopeROP>0 && slopeT>0
            UCSindicator(index,1)=16;
        elseif slopeWOB<0 && slopeROP==0 && slopeT==0
            UCSindicator(index,1)=17;
        elseif slopeWOB<0 && slopeROP>0 && slopeT==0
            UCSindicator(index,1)=18;
        elseif slopeWOB<0 && slopeROP==0 && slopeT>0
            UCSindicator(index,1)=19;
        elseif slopeWOB<0 && slopeROP<0 && slopeT>0
            UCSindicator(index,1)=20;
        elseif slopeWOB<0 && slopeROP>0 && slopeT<0
            UCSindicator(index,1)=21;
        elseif slopeWOB==0 && slopeROP>0 && slopeT>0
            UCSindicator(index,1)=22;
        elseif slopeWOB==0 && slopeROP<0 && slopeT<0
            UCSindicator(index,1)=23;
        elseif slopeWOB==0 && slopeROP==0 && slopeT<0
            UCSindicator(index,1)=24;
        elseif slopeWOB==0 && slopeROP>0 && slopeT==0
            UCSindicator(index,1)=25;
        elseif slopeWOB==0 && slopeROP==0 && slopeT>0
            UCSindicator(index,1)=26;
        elseif slopeWOB==0 && slopeROP<0 && slopeT==0
            UCSindicator(index,1)=27;
        else
            UCSindicator(index,1)=0; %potentially abnormal, unlikely this will
happen
        end

        %%Bin using smooth data
```

```
        index=find(B(:,1)==j);
        binWOB=Pad8Curve(index,74);
        firstWOB=binWOB(1:5); %takes first and last 5 points to average
        avgfirstWOB=mean(firstWOB);
        lastWOB=binWOB(end-4:end);
        avglastWOB=mean(lastWOB);
        slopeWOB=avglastWOB-avgfirstWOB;
        if slopeWOB>-W && slopeWOB<W %Redefine low slopes as zero slope
(constant)
            slopeWOB=0;
        end
        sWOBslope(index,1)=slopeWOB;
        if slopeWOB>0
            WOBslopeclasssm(index,1)=1;
        elseif slopeWOB<0
            WOBslopeclasssm(index,1)=-1;
        else
            WOBslopeclasssm(index,1)=0;
        end

        binROP=Pad8Curve(index,73);
        firstROP=binROP(1:5);
        avgfirstROP=mean(firstROP);
        lastROP=binROP(end-4:end);
        avglastROP=mean(lastROP);
        slopeROP=avglastROP-avgfirstROP;
        if slopeROP>-R && slopeROP<R %Redefine low slopes as zero slope
(constant)
            slopeROP=0;
        end
        sROPslope(index,1)=slopeROP;
        if slopeROP>0
            ROPslopeclasssm(index,1)=1;
        elseif slopeROP<0
            ROPslopeclasssm(index,1)=-1;
        else
            ROPslopeclasssm(index,1)=0;
        end

        binT=Pad8Curve(index,72);
        firstT=binT(1:5);
        avgfirstT=mean(firstT);
        lastT=binT(end-4:end);
        avglastT=mean(lastT);
        slopeT=avglastT-avgfirstT;
        if slopeT>-T && slopeT<T %Redefine low slopes as zero slope (constant)
```

```
        slopeT=0;
    end
    sTslope(index,1)=slopeT;
    if slopeT>0
        Tslopeclasssm(index,1)=1;
    elseif slopeT<0
        Tslopeclasssm(index,1)=-1;
    else
        Tslopeclasssm(index,1)=0;
    end

    %All possible parameter slope combinations
    if slopeWOB>0 && slopeROP>0 && slopeT>0
        sUCSindicator(index,1)=1;
    elseif slopeWOB<0 && slopeROP<0 && slopeT<0
        sUCSindicator(index,1)=2;
    elseif slopeWOB>0 && slopeROP==0 && slopeT>0
        sUCSindicator(index,1)=3;
    elseif slopeWOB<0 && slopeROP==0 && slopeT<0
        sUCSindicator(index,1)=4;
    elseif slopeWOB<0 && slopeROP<0 && slopeT==0
        sUCSindicator(index,1)=5;
    elseif slopeWOB>0 && slopeROP>0 && slopeT==0
        sUCSindicator(index,1)=6;
    elseif slopeWOB==0 && slopeROP==0 && slopeT==0
        sUCSindicator(index,1)=7;
    elseif slopeWOB==0 && slopeROP<0 && slopeT>0
        sUCSindicator(index,1)=8;
    elseif slopeWOB==0 && slopeROP>0 && slopeT<0
        sUCSindicator(index,1)=9;
    elseif slopeWOB>0 && slopeROP<0 && slopeT<0
        sUCSindicator(index,1)=10;
    elseif slopeWOB>0 && slopeROP==0 && slopeT==0
        sUCSindicator(index,1)=11;
    elseif slopeWOB>0 && slopeROP<0 && slopeT==0
        sUCSindicator(index,1)=12;
    elseif slopeWOB>0 && slopeROP==0 && slopeT<0
        sUCSindicator(index,1)=13;
    elseif slopeWOB>0 && slopeROP<0 && slopeT>0
        sUCSindicator(index,1)=14;
    elseif slopeWOB>0 && slopeROP>0 && slopeT<0
        sUCSindicator(index,1)=15;
    elseif slopeWOB<0 && slopeROP>0 && slopeT>0
        sUCSindicator(index,1)=16;
    elseif slopeWOB<0 && slopeROP==0 && slopeT==0
        sUCSindicator(index,1)=17;
```

```
        elseif slopeWOB<0 && slopeROP>0 && slopeT==0
            sUCSindicator(index,1)=18;
        elseif slopeWOB<0 && slopeROP==0 && slopeT>0
            sUCSindicator(index,1)=19;
        elseif slopeWOB<0 && slopeROP<0 && slopeT>0
            sUCSindicator(index,1)=20;
        elseif slopeWOB<0 && slopeROP>0 && slopeT<0
            sUCSindicator(index,1)=21;
        elseif slopeWOB==0 && slopeROP>0 && slopeT>0
            sUCSindicator(index,1)=22;
        elseif slopeWOB==0 && slopeROP<0 && slopeT<0
            sUCSindicator(index,1)=23;
        elseif slopeWOB==0 && slopeROP==0 && slopeT<0
            sUCSindicator(index,1)=24;
        elseif slopeWOB==0 && slopeROP>0 && slopeT==0
            sUCSindicator(index,1)=25;
        elseif slopeWOB==0 && slopeROP==0 && slopeT>0
            sUCSindicator(index,1)=26;
        elseif slopeWOB==0 && slopeROP<0 && slopeT==0
            sUCSindicator(index,1)=27;
        else
            sUCSindicator(index,1)=0; %potentially abnormal, unlikely this will
happen
        end

    end
end
if binpoints<10 && binpoints>=4
    for j=1:bins-1
        %%Bin using raw data
        index=find(B(:,1)==j);
        binWOB=Pad8Curve(index,2);
        firstWOB=binWOB(1:2); %takes first and last 2 points to average and get
slope
        avgfirstWOB=mean(firstWOB);
        lastWOB=binWOB(end-1:end);
        avglastWOB=mean(lastWOB);
        slopeWOB=avglastWOB-avgfirstWOB;
        if slopeWOB>-W && slopeWOB<W %Redefine low slopes as zero slope
(constant)
            slopeWOB=0;
        end
        WOBslope(index,1)=slopeWOB;
        if slopeWOB>0
            WOBslopeclass(index,1)=1;
        elseif slopeWOB<0
```

```
            WOBslopeclass(index,1)=-1;
        else
            WOBslopeclass(index,1)=0;
        end

        binROP=Pad8Curve(index,5);
        firstROP=binROP(1:2);
        avgfirstROP=mean(firstROP);
        lastROP=binROP(end-1:end);
        avglastROP=mean(lastROP);
        slopeROP=avglastROP-avgfirstROP;
        if slopeROP>-R && slopeROP<R %Redefine low slopes as zero slope
(constant)
            slopeROP=0;
        end
        ROPslope(index,1)=slopeROP;
        if slopeROP>0
            ROPslopeclass(index,1)=1;
        elseif slopeROP<0
            ROPslopeclass(index,1)=-1;
        else
            ROPslopeclass(index,1)=0;
        end

        binT=Pad8Curve(index,31);
        firstT=binT(1:2);
        avgfirstT=mean(firstT);
        lastT=binT(end-1:end);
        avglastT=mean(lastT);
        slopeT=avglastT-avgfirstT;
        if slopeT>-T && slopeT<T %Redefine low slopes as zero slope (constant)
            slopeT=0;
        end
        Tslope(index,1)=slopeT;
        if slopeT>0
            Tslopeclass(index,1)=1;
        elseif slopeT<0
            Tslopeclass(index,1)=-1;
        else
            Tslopeclass(index,1)=0;
        end

           %All possible parameter slope combinations
        if slopeWOB>0 && slopeROP>0 && slopeT>0
            UCSindicator(index,1)=1;
        elseif slopeWOB<0 && slopeROP<0 && slopeT<0
```

```
            UCSindicator(index,1)=2;
        elseif slopeWOB>0 && slopeROP==0 && slopeT>0
            UCSindicator(index,1)=3;
        elseif slopeWOB<0 && slopeROP==0 && slopeT<0
            UCSindicator(index,1)=4;
        elseif slopeWOB<0 && slopeROP<0 && slopeT==0
            UCSindicator(index,1)=5;
        elseif slopeWOB>0 && slopeROP>0 && slopeT==0
            UCSindicator(index,1)=6;
        elseif slopeWOB==0 && slopeROP==0 && slopeT==0
            UCSindicator(index,1)=7;
        elseif slopeWOB==0 && slopeROP<0 && slopeT>0
            UCSindicator(index,1)=8;
        elseif slopeWOB==0 && slopeROP>0 && slopeT<0
            UCSindicator(index,1)=9;
        elseif slopeWOB>0 && slopeROP<0 && slopeT<0
            UCSindicator(index,1)=10;
        elseif slopeWOB>0 && slopeROP==0 && slopeT==0
            UCSindicator(index,1)=11;
        elseif slopeWOB>0 && slopeROP<0 && slopeT==0
            UCSindicator(index,1)=12;
        elseif slopeWOB>0 && slopeROP==0 && slopeT<0
            UCSindicator(index,1)=13;
        elseif slopeWOB>0 && slopeROP<0 && slopeT>0
            UCSindicator(index,1)=14;
        elseif slopeWOB>0 && slopeROP>0 && slopeT<0
            UCSindicator(index,1)=15;
        elseif slopeWOB<0 && slopeROP>0 && slopeT>0
            UCSindicator(index,1)=16;
        elseif slopeWOB<0 && slopeROP==0 && slopeT==0
            UCSindicator(index,1)=17;
        elseif slopeWOB<0 && slopeROP>0 && slopeT==0
            UCSindicator(index,1)=18;
        elseif slopeWOB<0 && slopeROP==0 && slopeT>0
            UCSindicator(index,1)=19;
        elseif slopeWOB<0 && slopeROP<0 && slopeT>0
            UCSindicator(index,1)=20;
        elseif slopeWOB<0 && slopeROP>0 && slopeT<0
            UCSindicator(index,1)=21;
        elseif slopeWOB==0 && slopeROP>0 && slopeT>0
            UCSindicator(index,1)=22;
        elseif slopeWOB==0 && slopeROP<0 && slopeT<0
            UCSindicator(index,1)=23;
        elseif slopeWOB==0 && slopeROP==0 && slopeT<0
            UCSindicator(index,1)=24;
        elseif slopeWOB==0 && slopeROP>0 && slopeT==0
```

```
            UCSindicator(index,1)=25;
        elseif slopeWOB==0 && slopeROP==0 && slopeT>0
            UCSindicator(index,1)=26;
        elseif slopeWOB==0 && slopeROP<0 && slopeT==0
            UCSindicator(index,1)=27;
        else
            UCSindicator(index,1)=0; %potentially abnormal, unlikely this will
happen
        end


        %%Bin using smooth data
        index=find(B(:,1)==j);
        binWOB=Pad8Curve(index,74);
        firstWOB=binWOB(1:2); %takes first and last 2 points to average and get
slope
        avgfirstWOB=mean(firstWOB);
        lastWOB=binWOB(end-1:end);
        avglastWOB=mean(lastWOB);
        slopeWOB=avglastWOB-avgfirstWOB;
        if slopeWOB>-W && slopeWOB<W %Redefine low slopes as zero slope
(constant)
            slopeWOB=0;
        end
        sWOBslope(index,1)=slopeWOB;
        if slopeWOB>0
            WOBslopeclasssm(index,1)=1;
        elseif slopeWOB<0
            WOBslopeclasssm(index,1)=-1;
        else
            WOBslopeclasssm(index,1)=0;
        end

        binROP=Pad8Curve(index,73);
        firstROP=binROP(1:2);
        avgfirstROP=mean(firstROP);
        lastROP=binROP(end-1:end);
        avglastROP=mean(lastROP);
        slopeROP=avglastROP-avgfirstROP;
        if slopeROP>-R && slopeROP<R %Redefine low slopes as zero slope
(constant)
            slopeROP=0;
        end
        sROPslope(index,1)=slopeROP;
        if slopeROP>0
            ROPslopeclasssm(index,1)=1;
```

```
elseif slopeROP<0
    ROPslopeclasssm(index,1)=-1;
else
    ROPslopeclasssm(index,1)=0;
end

binT=Pad8Curve(index,72);
firstT=binT(1:2);
avgfirstT=mean(firstT);
lastT=binT(end-1:end);
avglastT=mean(lastT);
slopeT=avglastT-avgfirstT;
if slopeT>-T && slopeT<T %Redefine low slopes as zero slope (constant)
    slopeT=0;
end
sTslope(index,1)=slopeT;
if slopeT>0
    Tslopeclasssm(index,1)=1;
elseif slopeT<0
    Tslopeclasssm(index,1)=-1;
else
    Tslopeclasssm(index,1)=0;
end

%All possible parameter slope combinations
if slopeWOB>0 && slopeROP>0 && slopeT>0
    sUCSindicator(index,1)=1;
elseif slopeWOB<0 && slopeROP<0 && slopeT<0
    sUCSindicator(index,1)=2;
elseif slopeWOB>0 && slopeROP==0 && slopeT>0
    sUCSindicator(index,1)=3;
elseif slopeWOB<0 && slopeROP==0 && slopeT<0
    sUCSindicator(index,1)=4;
elseif slopeWOB<0 && slopeROP<0 && slopeT==0
    sUCSindicator(index,1)=5;
elseif slopeWOB>0 && slopeROP>0 && slopeT==0
    sUCSindicator(index,1)=6;
elseif slopeWOB==0 && slopeROP==0 && slopeT==0
    sUCSindicator(index,1)=7;
elseif slopeWOB==0 && slopeROP<0 && slopeT>0
    sUCSindicator(index,1)=8;
elseif slopeWOB==0 && slopeROP>0 && slopeT<0
    sUCSindicator(index,1)=9;
elseif slopeWOB>0 && slopeROP<0 && slopeT<0
    sUCSindicator(index,1)=10;
elseif slopeWOB>0 && slopeROP==0 && slopeT==0
```

```
            sUCSindicator(index,1)=11;
        elseif slopeWOB>0 && slopeROP<0 && slopeT==0
            sUCSindicator(index,1)=12;
        elseif slopeWOB>0 && slopeROP==0 && slopeT<0
            sUCSindicator(index,1)=13;
        elseif slopeWOB>0 && slopeROP<0 && slopeT>0
            sUCSindicator(index,1)=14;
        elseif slopeWOB>0 && slopeROP>0 && slopeT<0
            sUCSindicator(index,1)=15;
        elseif slopeWOB<0 && slopeROP>0 && slopeT>0
            sUCSindicator(index,1)=16;
        elseif slopeWOB<0 && slopeROP==0 && slopeT==0
            sUCSindicator(index,1)=17;
        elseif slopeWOB<0 && slopeROP>0 && slopeT==0
            sUCSindicator(index,1)=18;
        elseif slopeWOB<0 && slopeROP==0 && slopeT>0
            sUCSindicator(index,1)=19;
        elseif slopeWOB<0 && slopeROP<0 && slopeT>0
            sUCSindicator(index,1)=20;
        elseif slopeWOB<0 && slopeROP>0 && slopeT<0
            sUCSindicator(index,1)=21;
        elseif slopeWOB==0 && slopeROP>0 && slopeT>0
            sUCSindicator(index,1)=22;
        elseif slopeWOB==0 && slopeROP<0 && slopeT<0
            sUCSindicator(index,1)=23;
        elseif slopeWOB==0 && slopeROP==0 && slopeT<0
            sUCSindicator(index,1)=24;
        elseif slopeWOB==0 && slopeROP>0 && slopeT==0
            sUCSindicator(index,1)=25;
        elseif slopeWOB==0 && slopeROP==0 && slopeT>0
            sUCSindicator(index,1)=26;
        elseif slopeWOB==0 && slopeROP<0 && slopeT==0
            sUCSindicator(index,1)=27;
        else
            sUCSindicator(index,1)=0; %potentially abnormal, unlikely this will
happen
        end

    end
end
if binpoints<=3
    for j=1:bins-1
        %%Bin using raw data
        index=find(B(:,1)==j);
        binWOB=Pad8Curve(index,2);
```

```
        firstWOB=binWOB(1); %takes first point and last point and then takes
slope betwee
        lastWOB=binWOB(end);
        slopeWOB=lastWOB-firstWOB;
        if slopeWOB>-W && slopeWOB<W %Redefine low slopes as zero slope
(constant)
            slopeWOB=0;
        end
        WOBslope(index,1)=slopeWOB;
        if slopeWOB>0
            WOBslopeclass(index,1)=1;
        elseif slopeWOB<0
            WOBslopeclass(index,1)=-1;
        else
            WOBslopeclass(index,1)=0;
        end

        binROP=Pad8Curve(index,5);
        firstROP=binROP(1);
        lastROP=binROP(end);
        slopeROP=lastROP-firstROP;
        if slopeROP>-R && slopeROP<R %Redefine low slopes as zero slope
(constant)
            slopeROP=0;
        end
        ROPslope(index,1)=slopeROP;
        if slopeROP>0
            ROPslopeclass(index,1)=1;
        elseif slopeROP<0
            ROPslopeclass(index,1)=-1;
        else
            ROPslopeclass(index,1)=0;
        end

        binT=Pad8Curve(index,31);
        firstT=binT(1);
        lastT=binT(end);
        slopeT=lastT-firstT;
        if slopeT>-T && slopeT<T %Redefine low slopes as zero slope (constant)
            slopeT=0;
        end
        Tslope(index,1)=slopeT;
        if slopeT>0
            Tslopeclass(index,1)=1;
        elseif slopeT<0
            Tslopeclass(index,1)=-1;
```

```
        else
            Tslopeclass(index,1)=0;
        end

           %All possible parameter slope combinations
        if slopeWOB>0 && slopeROP>0 && slopeT>0
            UCSindicator(index,1)=1;
        elseif slopeWOB<0 && slopeROP<0 && slopeT<0
            UCSindicator(index,1)=2;
        elseif slopeWOB>0 && slopeROP==0 && slopeT>0
            UCSindicator(index,1)=3;
        elseif slopeWOB<0 && slopeROP==0 && slopeT<0
            UCSindicator(index,1)=4;
        elseif slopeWOB<0 && slopeROP<0 && slopeT==0
            UCSindicator(index,1)=5;
        elseif slopeWOB>0 && slopeROP>0 && slopeT==0
            UCSindicator(index,1)=6;
        elseif slopeWOB==0 && slopeROP==0 && slopeT==0
            UCSindicator(index,1)=7;
        elseif slopeWOB==0 && slopeROP<0 && slopeT>0
            UCSindicator(index,1)=8;
        elseif slopeWOB==0 && slopeROP>0 && slopeT<0
            UCSindicator(index,1)=9;
        elseif slopeWOB>0 && slopeROP<0 && slopeT<0
            UCSindicator(index,1)=10;
        elseif slopeWOB>0 && slopeROP==0 && slopeT==0
            UCSindicator(index,1)=11;
        elseif slopeWOB>0 && slopeROP<0 && slopeT==0
            UCSindicator(index,1)=12;
        elseif slopeWOB>0 && slopeROP==0 && slopeT<0
            UCSindicator(index,1)=13;
        elseif slopeWOB>0 && slopeROP<0 && slopeT>0
            UCSindicator(index,1)=14;
        elseif slopeWOB>0 && slopeROP>0 && slopeT<0
            UCSindicator(index,1)=15;
        elseif slopeWOB<0 && slopeROP>0 && slopeT>0
            UCSindicator(index,1)=16;
        elseif slopeWOB<0 && slopeROP==0 && slopeT==0
            UCSindicator(index,1)=17;
        elseif slopeWOB<0 && slopeROP>0 && slopeT==0
            UCSindicator(index,1)=18;
        elseif slopeWOB<0 && slopeROP==0 && slopeT>0
            UCSindicator(index,1)=19;
        elseif slopeWOB<0 && slopeROP<0 && slopeT>0
            UCSindicator(index,1)=20;
        elseif slopeWOB<0 && slopeROP>0 && slopeT<0
```

```
            UCSindicator(index,1)=21;
        elseif slopeWOB==0 && slopeROP>0 && slopeT>0
            UCSindicator(index,1)=22;
        elseif slopeWOB==0 && slopeROP<0 && slopeT<0
            UCSindicator(index,1)=23;
        elseif slopeWOB==0 && slopeROP==0 && slopeT<0
            UCSindicator(index,1)=24;
        elseif slopeWOB==0 && slopeROP>0 && slopeT==0
            UCSindicator(index,1)=25;
        elseif slopeWOB==0 && slopeROP==0 && slopeT>0
            UCSindicator(index,1)=26;
        elseif slopeWOB==0 && slopeROP<0 && slopeT==0
            UCSindicator(index,1)=27;
        else
            UCSindicator(index,1)=0; %potentially abnormal, unlikely this will
happen
        end

        %%Bin using smooth data
        index=find(B(:,1)==j);
        binWOB=Pad8Curve(index,2);
        firstWOB=binWOB(1); %takes first point and last point and then takes
slope betwee
        lastWOB=binWOB(end);
        slopeWOB=lastWOB-firstWOB;
        if slopeWOB>-W && slopeWOB<W %Redefine low slopes as zero slope
(constant)
            slopeWOB=0;
        end
        sWOBslope(index,1)=slopeWOB;
        if slopeWOB>0
            WOBslopeclasssm(index,1)=1;
        elseif slopeWOB<0
            WOBslopeclasssm(index,1)=-1;
        else
            WOBslopeclasssm(index,1)=0;
        end

        binROP=Pad8Curve(index,5);
        firstROP=binROP(1);
        lastROP=binROP(end);
        slopeROP=lastROP-firstROP;
        if slopeROP>-R && slopeROP<R %Redefine low slopes as zero slope
(constant)
            slopeROP=0;
        end
```

```
        sROPslope(index,1)=slopeROP;
        if slopeROP>0
            ROPslopeclasssm(index,1)=1;
        elseif slopeROP<0
            ROPslopeclasssm(index,1)=-1;
        else
            ROPslopeclasssm(index,1)=0;
        end

        binT=Pad8Curve(index,31);
        firstT=binT(1);
        lastT=binT(end);
        slopeT=lastT-firstT;
        if slopeT>-T && slopeT<T %Redefine low slopes as zero slope (constant)
            slopeT=0;
        end
        sTslope(index,1)=slopeT;
        if slopeT>0
            Tslopeclasssm(index,1)=1;
        elseif slopeT<0
            Tslopeclasssm(index,1)=-1;
        else
            Tslopeclasssm(index,1)=0;
        end

        %All possible parameter slope combinations
        if slopeWOB>0 && slopeROP>0 && slopeT>0
            sUCSindicator(index,1)=1;
        elseif slopeWOB<0 && slopeROP<0 && slopeT<0
            sUCSindicator(index,1)=2;
        elseif slopeWOB>0 && slopeROP==0 && slopeT>0
            sUCSindicator(index,1)=3;
        elseif slopeWOB<0 && slopeROP==0 && slopeT<0
            sUCSindicator(index,1)=4;
        elseif slopeWOB<0 && slopeROP<0 && slopeT==0
            sUCSindicator(index,1)=5;
        elseif slopeWOB>0 && slopeROP>0 && slopeT==0
            sUCSindicator(index,1)=6;
        elseif slopeWOB==0 && slopeROP==0 && slopeT==0
            sUCSindicator(index,1)=7;
        elseif slopeWOB==0 && slopeROP<0 && slopeT>0
            sUCSindicator(index,1)=8;
        elseif slopeWOB==0 && slopeROP>0 && slopeT<0
            sUCSindicator(index,1)=9;
        elseif slopeWOB>0 && slopeROP<0 && slopeT<0
            sUCSindicator(index,1)=10;
```

```
        elseif slopeWOB>0 && slopeROP==0 && slopeT==0
            sUCSindicator(index,1)=11;
        elseif slopeWOB>0 && slopeROP<0 && slopeT==0
            sUCSindicator(index,1)=12;
        elseif slopeWOB>0 && slopeROP==0 && slopeT<0
            sUCSindicator(index,1)=13;
        elseif slopeWOB>0 && slopeROP<0 && slopeT>0
            sUCSindicator(index,1)=14;
        elseif slopeWOB>0 && slopeROP>0 && slopeT<0
            sUCSindicator(index,1)=15;
        elseif slopeWOB<0 && slopeROP>0 && slopeT>0
            sUCSindicator(index,1)=16;
        elseif slopeWOB<0 && slopeROP==0 && slopeT==0
            sUCSindicator(index,1)=17;
        elseif slopeWOB<0 && slopeROP>0 && slopeT==0
            sUCSindicator(index,1)=18;
        elseif slopeWOB<0 && slopeROP==0 && slopeT>0
            sUCSindicator(index,1)=19;
        elseif slopeWOB<0 && slopeROP<0 && slopeT>0
            sUCSindicator(index,1)=20;
        elseif slopeWOB<0 && slopeROP>0 && slopeT<0
            sUCSindicator(index,1)=21;
        elseif slopeWOB==0 && slopeROP>0 && slopeT>0
            sUCSindicator(index,1)=22;
        elseif slopeWOB==0 && slopeROP<0 && slopeT<0
            sUCSindicator(index,1)=23;
        elseif slopeWOB==0 && slopeROP==0 && slopeT<0
            sUCSindicator(index,1)=24;
        elseif slopeWOB==0 && slopeROP>0 && slopeT==0
            sUCSindicator(index,1)=25;
        elseif slopeWOB==0 && slopeROP==0 && slopeT>0
            sUCSindicator(index,1)=26;
        elseif slopeWOB==0 && slopeROP<0 && slopeT==0
            sUCSindicator(index,1)=27;
        else
            sUCSindicator(index,1)=0; %potentially abnormal, unlikely this will
happen
        end

    end
end

IDmatrix=sUCSindicator;
SlopeMatrix=[WOBslope ROPslope Tslope WOBslopeclass ROPslopeclass Tslopeclass
sWOBslope sROPslope sTslope  WOBslopeclasssm ROPslopeclasssm Tslopeclasssm];
```

```
clear slopeWOB slopeROP slopeT index binT firstT lastT binROP firstROP lastROP
binWOB firstWOB lastWOB
```

R. Second Column of the CBM: Number of Constant Parameters

The program presented below is referred to in the CBM master program in Appendix P as “constantparam.m”.

```
%Create second column of identity matrix: identify how many parameters are
constant

numberconst=zeros(m,1);
for j=1:bins-1
    %Bin using raw data
    index=find(B(:,1)==j);
    WOBslope=sWOBslope(index,1);
    if WOBslope(1)==0
        a=1;
    else
        a=0;
    end
    ROPslope=sROPslope(index,1);
    if ROPslope(1)==0
        b=1;
    else
        b=0;
    end
    Tslope=sTslope(index,1);
    if Tslope(1)==0
        c=1;
    else
        c=0;
    end
    total=a+b+c;
    if total==1
        numberconst(index,1)=1; %1 constant
    elseif total==2
        numberconst(index,1)=2; %2 constant
    elseif total==3
        numberconst(index,1)=3; %all constant
    elseif total==0
        numberconst(index,1)=0; %no constant
```

```
    end
end

IDmatrix=[IDmatrix numberconst];

clear total Tslope ROPslope WOBslope a b c j index
```

S. Third Column of the CBM: Define Magnitude Change Classification Scheme

The program presented below is referred to in the CBM master program in Appendix P as “mag.m”.

```
%Define magnitude change indentifiers for each bin using smoothed data slopes
magchangeWOB=zeros(m,1);
magchangeROP=zeros(m,1);
magchangeT=zeros(m,1);
for j=1:bins-1
        index=find(B(:,1)==j);
        sWOB=sWOBslope(index,1);
        sROP=sROPslope(index,1);
        sT=sTslope(index,1);
        if sWOB(1)>5
            magchangeWOB(index,1)=2;
        elseif sWOB(1)<=5 && sWOB(1)>0
            magchangeWOB(index,1)=1;
        else
            magchangeWOB(index,1)=0;
        end
        if sWOB(1)<-5
            magchangeWOB(index,1)=2;
        elseif sWOB(1)>=-5 && sWOB(1)<0
            magchangeWOB(index,1)=1;
        else
            magchangeWOB(index,1)=0;
        end

        if sROP(1)>15
            magchangeROP(index,1)=2;
        elseif sROP(1)<=15 && sROP(1)>0
            magchangeROP(index,1)=1;
        else
```

```
            magchangeROP(index,1)=0;
        end
         if sROP(1)<-15
            magchangeROP(index,1)=2;
        elseif sROP(1)>=-15 && sROP(1)<0
            magchangeROP(index,1)=1;
        else
            magchangeROP(index,1)=0;
         end

        if sT(1)>2000
            magchangeT(index,1)=2;
        elseif sT(1)<=2000 && sT(1)>0
            magchangeT(index,1)=1;
        else
            magchangeT(index,1)=0;
        end
         if sT(1)<-2000
            magchangeT(index,1)=2;
        elseif sT(1)>=-2000 && sT(1)<0
            magchangeT(index,1)=1;
        else
            magchangeT(index,1)=0;
        end
end

clear sT sROP sWOB index
```

T. THIRD COLUMN OF THE CBM: ASSIGN SLOPE MAGNITUDE CHANGE CLASSIFICATION

The program presented below is referred to in the CBM master program in Appendix P as “slopemagnitude.m”.

```
%Create 3rd column in identity matrix: identify which line of characteristic
set of possibilities (based on # of constant parameters) corresponds with the
magnitude change combination

magnitudechange=zeros(m,1);
for j=1:bins-1
```

```
index=find(B(:,1)==j);
numberconstant=numberconst(index,1);
WOBmag=magchangeWOB(index,1);
ROPmag=magchangeROP(index,1);
Tmag=magchangeT(index,1);
%Define which parameters are non-constant (always in WOB, ROP, T order)
if WOBmag(1)>0
    a=WOBmag;
elseif ROPmag(1)>0
    a=ROPmag;
elseif Tmag(1)>0
    a=Tmag;
else
    a=0;
end
if a(1)>0
    if ROPmag(1)>0
        b=ROPmag;
    elseif Tmag(1)>0
        b=Tmag;
    else
        b=0;
    end
else
    b=0;
end
if b(1)>0
    if Tmag(1)>0
        c=Tmag;
    else
        c=0;
    end
else
    c=0;
end

%Define characteristic set of parameter combinations given number constant
if numberconstant(1)==0 %No constant parameters
    if a(1)==1 && b(1)==1 && c(1)==1
        magnitudechange(index,1)=1;
    elseif a(1)==1 && b(1)==1 && c(1)==2
        magnitudechange(index,1)=2;
    elseif a(1)==1 && b(1)==2 && c(1)==1
        magnitudechange(index,1)=3;
    elseif a(1)==1 && b(1)==2 && c(1)==2
        magnitudechange(index,1)=4;
```

```
        elseif a(1)==2 && b(1)==1 && c(1)==1
            magnitudechange(index,1)=5;
        elseif a(1)==2 && b(1)==1 && c(1)==2
            magnitudechange(index,1)=6;
        elseif a(1)==2 && b(1)==2 && c(1)==1
            magnitudechange(index,1)=7;
        elseif a(1)==2 && b(1)==2 && c(1)==2
            magnitudechange(index,1)=8;
        end
    elseif numberconstant(1)==1 %One constant parameter
        if a(1)==1 && b(1)==1
            magnitudechange(index,1)=1;
        elseif a(1)==1 && b(1)==2
            magnitudechange(index,1)=2;
        elseif a(1)==2 && b(1)==1
            magnitudechange(index,1)=3;
        elseif a(1)==2 && b(1)==2
            magnitudechange(index,1)=4;
        end
    elseif numberconstant(1)==2 %Two constant parameters
        if a(1)==1
            magnitudechange(index,1)=1;
        elseif a(1)==2
            magnitudechange(index,1)=2;
        end
    elseif numberconstant(1)==3
            magnitudechange(index,1)=0;
    end

end

IDmatrix=[IDmatrix magnitudechange];

clear a b c numberconstant ROPmag Tmag WOBmag index
```

U. USE CBM TO STATISTICALLY ASSIGN UCS SLOPE PREDICTION

The program presented below is referred to in the Well 8 curve processing program in Appendix L as “LikelyUCS.m”.

```
%Use CBM to statistically assign a UCS slope prediction

%%Bin UCS slope
binnedUCSslope=zeros(m,1);
 for j=1:bins-1
    index=find(B(:,1)==j);
    binUCS=Pad8Curve(index,99);
    slope=mode(binUCS);
    binnedUCSslope(index,1)=slope;
end

clear binUCS slope index j


%%Split 1st layer of ID matrix into groups and assign UCS slopes



[Groups1,ID1]=findgroups(IDmatrix(:,1));
Identities1=[ID1];
numberofgroups=max(Groups1);
PredictedUCS1stlayer=zeros(m,1);
IdentitySlopePrediction1=zeros(numberofgroups,1);
for i=1:numberofgroups
    index=find(Groups1(:,1)==i);
    UCSvalues=Pad8Curve(index,99);
    [X,F]=mode(UCSvalues);
    X1index=find(UCSvalues>X);
    X2index=find(UCSvalues<X);
    n=numel(UCSvalues);
    if F/n>=.5
        UCSvalues(X1index,1)=2;
        UCSvalues(X2index,1)=2;
        PredictedUCS1stlayer(index,1)=UCSvalues;
        IdentitySlopePrediction1(i,1)=X;
    else
        UCSvalues(:,1)=2;
        PredictedUCS1stlayer(index,1)=UCSvalues;
        IdentitySlopePrediction1(i,1)=2;
    end
end
Identities1=[Identities1 IdentitySlopePrediction1];
clear index UCSvalues X1index X2index F n X

%%Split first and second layers of ID Matrix into groups and assign UCS slopes
```

```
[Groups2,ID11,ID22,ID33]=findgroups(IDmatrix(:,1),IDmatrix(:,2),IDmatrix(:,3));
Identities2=[ID11 ID22 ID33];
numberofgroups=max(Groups2);
PredictedUCS2ndlayer=zeros(m,1);
PredictedUCS=PredictedUCS1stlayer;
IdentitySlopePrediction2=zeros(numberofgroups,1);
for i=1:numberofgroups
    index=find(Groups2(:,1)==i);
    UCSvalues=Pad8Curve(index,99);
    UCSvalues2=UCSvalues;
    [X,F]=mode(UCSvalues);
    X1index=find(UCSvalues>X);
    X2index=find(UCSvalues<X);
    n=numel(UCSvalues);
    if F/n>=.5
        UCSvalues2(X1index,1)=2;
        UCSvalues2(X2index,1)=2;
        PredictedUCS2ndlayer(index,1)=UCSvalues2;
        UCSvalues(X1index,1)=PredictedUCS1stlayer(X1index,1);
        UCSvalues(X2index,1)=PredictedUCS1stlayer(X2index,1);
        PredictedUCS(index,1)=UCSvalues;
        IdentitySlopePrediction2(i,2)=X;
    else
        UCSvalues2(:,1)=2;
        PredictedUCS2ndlayer(index,1)=UCSvalues2;
    end
end
Identities2=[Identities2 IdentitySlopePrediction2];
clear index UCSvalues X1index X2index F n X
```

V. CREATE PARAMETER SLOPE FEATURES FOR RANDOM FOREST

The program presented below is referred to in the Well 8 curve processing program in Appendix L as “DataLabels.m”.

```
%Assign classifications to parameter slope features for use in Random Forest

%%Cut out non-drilling time and mark transitions from drilling time sections

%%Convert time to cumulative seconds (in 10 second intervals)
```

```
time=Pad8Curve(:,25);
timesec=time*60*60;

%%Extract smoothed parameters from data
smoothWOB=Pad8Curve(:,74);
smoothROP=Pad8Curve(:,73);
smoothT=Pad8Curve(:,72);
%smoothWOB=Pad8Curve(:,2); %use these 3 when want to use raw data
%smoothROP=Pad8Curve(:,5);
%smoothT=Pad8Curve(:,31);

%%Extract UCS derivative information
dUCS=Pad8Curve(:,79);
dUCSclass=Pad8Curve(:,99); %classification of UCS slope

%%Find breaks in data and converts first point of next drilling interval into
an indicator of a new interval (indicated by a 1)
n=numel(timesec);
timeindicator=zeros(n,1);
for i=2:n
    if timesec(i)-timesec(i-1)>10
        timeindicator(i)=1;
    end
end
timeindicator(1)=1;

%%Label data based on features for Random Forest Classification Instantaneous
Slope
instWOB=zeros(n,1);
instROP=zeros(n,1);
instT=zeros(n,1);
for i=2:n
    instWOB(i)=smoothWOB(i)-smoothWOB(i-1);
    instROP(i)=smoothROP(i)-smoothROP(i-1);
    instT(i)=smoothT(i)-smoothT(i-1);
end

%%Instantaneous Slope Character, Erratic Behavior, and Macro Slope Character
charWOB=zeros(n,1);
charROP=zeros(n,1);
charT=zeros(n,1);
classWOB=zeros(n,1);
classROP=zeros(n,1);
classT=zeros(n,1);
classUCS=zeros(n,1);
```

```
Rthreshold=2;    %Threshold for WOB range to indicate macro behavior as
constant
Wthreshold=.6;   %"" for ROP
Tthreshold=175;  %"" for Torque
oscillating=zeros(n,1);
    %Convert instantaneous slope data to classification.
    %This is the instantaneous slope character.
    %Each parameter instantaneous slope is scaled to a 1 2 3 .. scale then
classed by unit
    %interval
normWOBslope=instWOB*10;
normROPslope=instROP;
normTslope=instT/10;
normUCSslope=dUCS/100;
    for i=1:n
        if normWOBslope(i)>0 && normWOBslope(i)<=1
            classWOB(i)=1;
        elseif normWOBslope(i)>1 && normWOBslope(i)<=2
            classWOB(i)=2;
        elseif normWOBslope(i)>2 && normWOBslope(i)<=3
            classWOB(i)=3;
        elseif normWOBslope(i)>3 && normWOBslope(i)<=4
            classWOB(i)=4;
        elseif normWOBslope(i)>4 && normWOBslope(i)<=5
            classWOB(i)=5;
        elseif normWOBslope(i)>5 && normWOBslope(i)<=6
            classWOB(i)=6;
        elseif normWOBslope(i)>6 && normWOBslope(i)<=7
            classWOB(i)=7;
        elseif normWOBslope(i)>7 && normWOBslope(i)<=8
            classWOB(i)=8;
        elseif normWOBslope(i)>8 && normWOBslope(i)<=9
            classWOB(i)=9;
        elseif normWOBslope(i)>9 && normWOBslope(i)<=10
            classWOB(i)=10;
        elseif normWOBslope(i)>10
            classWOB(i)=11;
        elseif normWOBslope(i)<0 && normWOBslope(i)>=-1
            classWOB(i)=-1;
        elseif normWOBslope(i)<-1 && normWOBslope(i)>=-2
            classWOB(i)=-2;
        elseif normWOBslope(i)<-2 && normWOBslope(i)>=-3
            classWOB(i)=-3;
        elseif normWOBslope(i)<-3 && normWOBslope(i)>=-4
            classWOB(i)=-4;
        elseif normWOBslope(i)<-4 && normWOBslope(i)>=-5
```

```
        classWOB(i)=-5;
    elseif normWOBslope(i)<-5 && normWOBslope(i)>=-6
        classWOB(i)=-6;
    elseif normWOBslope(i)<-6 && normWOBslope(i)>=-7
        classWOB(i)=-7;
    elseif normWOBslope(i)<-7 && normWOBslope(i)>=-8
        classWOB(i)=-8;
    elseif normWOBslope(i)<-8 && normWOBslope(i)>=-9
        classWOB(i)=-9;
    elseif normWOBslope(i)<-9 && normWOBslope(i)>=-10
        classWOB(i)=-10;
    elseif normWOBslope(i)<-10
        classWOB(i)=-11;
    end

     if normROPslope(i)>0 && normROPslope(i)<=1
        classROP(i)=1;
    elseif normROPslope(i)>1 && normROPslope(i)<=2
        classROP(i)=2;
    elseif normROPslope(i)>2 && normROPslope(i)<=3
        classROP(i)=3;
    elseif normROPslope(i)>3 && normROPslope(i)<=4
        classROP(i)=4;
    elseif normROPslope(i)>4 && normROPslope(i)<=5
        classROP(i)=5;
    elseif normROPslope(i)>5 && normROPslope(i)<=6
        classROP(i)=6;
    elseif normROPslope(i)>6 && normROPslope(i)<=7
        classROP(i)=7;
    elseif normROPslope(i)>7 && normROPslope(i)<=8
        classROP(i)=8;
    elseif normROPslope(i)>8 && normROPslope(i)<=9
        classROP(i)=9;
    elseif normROPslope(i)>9 && normROPslope(i)<=10
        classROP(i)=10;
    elseif normROPslope(i)>10
        classROP(i)=11;
    elseif normROPslope(i)<0 && normROPslope(i)>=-1
        classROP(i)=-1;
    elseif normROPslope(i)<-1 && normROPslope(i)>=-2
        classROP(i)=-2;
    elseif normROPslope(i)<-2 && normROPslope(i)>=-3
        classROP(i)=-3;
    elseif normROPslope(i)<-3 && normROPslope(i)>=-4
        classROP(i)=-4;
    elseif normROPslope(i)<-4 && normROPslope(i)>=-5
```

```
        classROP(i)=-5;
    elseif normROPslope(i)<-5 && normROPslope(i)>=-6
        classROP(i)=-6;
    elseif normROPslope(i)<-6 && normROPslope(i)>=-7
        classROP(i)=-7;
    elseif normROPslope(i)<-7 && normROPslope(i)>=-8
        classROP(i)=-8;
    elseif normROPslope(i)<-8 && normROPslope(i)>=-9
        classROP(i)=-9;
    elseif normROPslope(i)<-9 && normROPslope(i)>=-10
        classROP(i)=-10;
    elseif normROPslope(i)<-10
        classROP(i)=-11;
    end


     if normTslope(i)>0 && normTslope(i)<=1
        classT(i)=1;
    elseif normTslope(i)>1 && normTslope(i)<=2
        classT(i)=2;
    elseif normTslope(i)>2 && normTslope(i)<=3
        classT(i)=3;
    elseif normTslope(i)>3 && normTslope(i)<=4
        classT(i)=4;
    elseif normTslope(i)>4 && normTslope(i)<=5
        classT(i)=5;
    elseif normTslope(i)>5 && normTslope(i)<=6
        classT(i)=6;
    elseif normTslope(i)>6 && normTslope(i)<=7
        classT(i)=7;
    elseif normTslope(i)>7 && normTslope(i)<=8
        classT(i)=8;
    elseif normTslope(i)>8 && normTslope(i)<=9
        classT(i)=9;
    elseif normTslope(i)>9 && normTslope(i)<=10
        classT(i)=10;
    elseif normTslope(i)>10
        classT(i)=11;
    elseif normTslope(i)<0 && normTslope(i)>=-1
        classT(i)=-1;
    elseif normTslope(i)<-1 && normTslope(i)>=-2
        classT(i)=-2;
    elseif normTslope(i)<-2 && normTslope(i)>=-3
        classT(i)=-3;
    elseif normTslope(i)<-3 && normTslope(i)>=-4
        classT(i)=-4;
```

```
elseif normTslope(i)<-4 && normTslope(i)>=-5
    classT(i)=-5;
elseif normTslope(i)<-5 && normTslope(i)>=-6
    classT(i)=-6;
elseif normTslope(i)<-6 && normTslope(i)>=-7
    classT(i)=-7;
elseif normTslope(i)<-7 && normTslope(i)>=-8
    classT(i)=-8;
elseif normTslope(i)<-8 && normTslope(i)>=-9
    classT(i)=-9;
elseif normTslope(i)<-9 && normTslope(i)>=-10
    classT(i)=-10;
elseif normTslope(i)<-10
    classT(i)=-11;
 end

 if normUCSslope(i)>0 && normUCSslope(i)<=2
    classUCS(i)=1;
%elseif normUCSslope(i)>1 && normUCSslope(i)<=2
    classUCS(i)=2;
elseif normUCSslope(i)>2 && normUCSslope(i)<=4
    classUCS(i)=3;
%elseif normUCSslope(i)>3 && normUCSslope(i)<=4
    classUCS(i)=4;
elseif normUCSslope(i)>4 && normUCSslope(i)<=6
    classUCS(i)=5;
%elseif normUCSslope(i)>5 && normUCSslope(i)<=6
    classUCS(i)=6;
elseif normUCSslope(i)>6 && normUCSslope(i)<=8
    classUCS(i)=7;
%elseif normUCSslope(i)>7 && normUCSslope(i)<=8
    classUCS(i)=8;
elseif normUCSslope(i)>8 && normUCSslope(i)<=10
    classUCS(i)=9;
%elseif normUCSslope(i)>9 && normUCSslope(i)<=10
    classUCS(i)=10;
elseif normUCSslope(i)>10
    classUCS(i)=11;
elseif normUCSslope(i)<0 && normUCSslope(i)>=-2
    classUCS(i)=-1;
%elseif normUCSslope(i)<-1 && normUCSslope(i)>=-2
    classUCS(i)=-2;
elseif normUCSslope(i)<-2 && normUCSslope(i)>=-4
    classUCS(i)=-3;
%elseif normUCSslope(i)<-3 && normUCSslope(i)>=-4
    classUCS(i)=-4;
```

```
        elseif normUCSslope(i)<-4 && normUCSslope(i)>=-6
            classUCS(i)=-5;
        %elseif normUCSslope(i)<-5 && normUCSslope(i)>=-6
            classUCS(i)=-6;
        elseif normUCSslope(i)<-6 && normUCSslope(i)>=-8
            classUCS(i)=-7;
        %elseif normUCSslope(i)<-7 && normUCSslope(i)>=-8
            classUCS(i)=-8;
        elseif normUCSslope(i)<-8 && normUCSslope(i)>=-10
            classUCS(i)=-9;
        %elseif normUCSslope(i)<-9 && normUCSslope(i)>=-10
            classUCS(i)=-10;
        elseif normUCSslope(i)<-10
            classUCS(i)=-11;
        end
    end
    %Using instantaneous slope character, look at data window to determine
    %macro slope character at each point
    window=4; %Number of data points on each side of current point to view
    for i=window+1:n-window
        timewindow=timeindicator(i-window:i+window); %Create a data window for
each parameter, in terms of the parameter values and their slope classification
        drillingbreak=find(timewindow==1);

        if drillingbreak < window+1
            start=window+1-drillingbreak; %Allows the program to index the
window to start at the start of the drilling interval
            %windowWOB=smoothWOB(i-start:i+window);
            %windowROP=smoothROP(i-start:i+window);
            %windowT=smoothT(i-start:i+window);
            windowWOB=smoothWOB(i-start:i+window);
            windowROP=smoothROP(i-start:i+window);
            windowT=smoothT(i-start:i+window);
            windowWOBclass=classWOB(i-start+1:i+window);%the first point in a
drilling interval will always have zero slope so want to exclude
            windowROPclass=classROP(i-start+1:i+window);
            windowTclass=classT(i-start+1:i+window);
        elseif drillingbreak > window+1
            start=drillingbreak-window+1; %Allows the program to index the
window to end at the start of the next drilling interval
            %windowWOB=smoothWOB(i-window:i+start-1);%Will end window in last
point of current drilling interval
            %windowROP=smoothROP(i-window:i+start-1);
            %windowT=smoothT(i-window:i+start-1);
            windowWOB=smoothWOB(i-window:i+start-1);%Will end window in last
point of current drilling interval
```

```
            windowROP=smoothROP(i-window:i+start-1);
            windowT=smoothT(i-window:i+start-1);
            windowWOBclass=classWOB(i-window:i+start-1);
            windowROPclass=classROP(i-window:i+start-1);
            windowTclass=classT(i-window:i+start-1);
        elseif drillingbreak == window+1
            windowWOB=0;
            windowROP=0;
            windowT=0;
            windowWOBclass=0;
            windowROPclass=0;
            windowTclass=0;
        else
            %windowWOB=smoothWOB(i-window:i+window);
            %windowROP=smoothROP(i-window:i+window);
            %windowT=smoothT(i-window:i+window);
            windowWOB=smoothWOB(i-window:i+window);
            windowROP=smoothROP(i-window:i+window);
            windowT=smoothT(i-window:i+window);
            windowWOBclass=classWOB(i-window:i+window);
            windowROPclass=classROP(i-window:i+window);
            windowTclass=classT(i-window:i+window);
         end
        %Evaluate the windows created above for range of actual values. If
        %range below a specified threshold specific to each parameter,
        %the character is evaluated as constant. Otherwise the character is
        %evaluated as the slope classification at that point.
        if windowWOB==0
            charWOB(i)=0;
        else
            WOBrange=max(windowWOB)-min(windowWOB);
            if WOBrange < Wthreshold
                charWOB(i)=0;
            else
                charWOB(i)=classWOB(i);
            end
        end
        if windowROP==0
            charROP(i)=0;
        else
            ROPrange=max(windowROP)-min(windowROP);
            if ROPrange < Rthreshold
                charROP(i)=0;
            else
                charROP(i)=classROP(i);
            end
```

```
        end
        if windowT==0
            charT(i)=0;
        else
            Trange=max(windowT)-min(windowT);
            if Trange < Tthreshold
                charT(i)=0;
            else
                charT(i)=classT(i);
            end
        end

    end


                %CHANGE THE BELOW LABELS TO smoothWOB ETC WHEN SWITCHING
                %BETWEEN RAW AND SMOOTHED DATA
Labels=[timesec smoothWOB smoothROP smoothT dUCS dUCSclass timeindicator
instWOB instROP instT classWOB classROP classT classUCS charWOB charROP charT];
Labelsadd=[timesec timeindicator instWOB instROP instT classWOB classROP classT
classUCS charWOB charROP charT];

HandClassification=Pad8Curve(:,98);
erraticWOB=Pad8Curve(:,53);
erraticROP=Pad8Curve(:,55);
erraticT=Pad8Curve(:,52);
Labels=[Labels erraticWOB erraticROP erraticT HandClassification];


LabelsTable=array2table(Labels);
LabelsTable.Properties.VariableNames={'TimeSeconds' 'SmoothWOB' 'SmoothROP'
'SmoothT' 'dUCS' 'dUCSclass' 'timeindicator' 'instantWOBslope'
'instantROPslope' 'instantTslope' 'instWOBclass' 'instROPclass' 'instTclass'
'UCSslopeclassification' 'WOBchar' 'ROPchar' 'Tchar' 'ErraticWOB' 'ErraticROP'
'ErraticT' 'HandClassification'};
writetable(LabelsTable,'C:\Users\Carolyn\Documents\Masters Research\Spring
2018\Apache\Processed Single Files\Labels.csv')
```

W. CREATE 1ST HALF AND 2ND HALF TEST AND TRAIN SETS FOR RANDOM FOREST

The program presented below will split the processed Well 8 curve data in half based on MD to create test and train sets for Random Forest analysis.

```
%Split Pad 8 Curve into 1st and 2nd half, create sub files

Pad8Curve=readtable('Pad8Curve.csv');
Pad8Curve=table2array(Pad8Curve);
max=Pad8Curve(end,1);
min=Pad8Curve(1,1);
span=max-min;
half=span/2;
Halfpoint=min+half;
Pad8Curvepresplit=Pad8Curve;
indexlowerdepths=find(Pad8Curvepresplit(:,1)>=Halfpoint);
Pad8Curve2ndhalf=Pad8Curvepresplit(indexlowerdepths,:);
indexupperdepths=find(Pad8Curvepresplit(:,1)<Halfpoint);
Pad8Curve1sthalf=Pad8Curvepresplit(indexupperdepths,:);



Pad8Curve1sthalfTable=array2table(Pad8Curve1sthalf);
Pad8Curve1sthalfTable.Properties.VariableNames={'HoleDepth' 'WeightonBit'
'BitRPM' 'DifferentialPressure' 'RateofPenetration' 'MotorRPM' 'BlockHeight'
'RotaryRPM' 'TotalPumpOutput' 'BitDepth' 'RotaryTorque' 'Inclination' 'Azimuth'
'DysfunctionDepth' 'BitBallingBelief' 'BitBounceBelief' 'StickSlipBelief'
'WhirlBelief' 'OtherDysfunctionBelief' 'NoDysfunctionBelief' 'UCSanalog' 'BHA'
'MSE' 'MSEwithRotaryTorque' 'Time' 'CalcedBitRPM' 'DrillingEfficiency'
'BitAggressiveness' 'HSI' 'Torque' 'TVDSurvey' 'CourseNorthSurvey'
'CourseEastSurvey' 'TVD' 'CourseNorth' 'CourseEast' 'BinnedMSE' 'SmoothMSE'
'dROP' 'dRPM' 'dTorque' 'dWOB' 'dTime' 'dsmoothROP' 'dsmoothRPM'
'dsmoothTorque' 'dsmoothWOB' 'dsmoothTime' 'sWOBandTvsROP' 'ZeroRatio'
'MicroErraticTorque' 'MicroErraticWOB' 'MicroErraticRPM' 'MicroErraticROP'
'TorqueMagnitudeChange' 'WOBMagnitudeChange' 'RPMMagnitudeChange'
'ROPMagnitudeChange' 'sWOBandTandROP' 'sWOBandROPvsT' 'sTandROPvsWOB'
'WOBandTandROP' 'WOBandROPvsT' 'TandROPvsWOB' 'WOBandTvsROP' 'ROPandWOB'
'CoefficientofFriction' 'dROPvdRPM' 'dROPvdT' 'DOC' 'smoothTorque' 'smoothROP'
'smoothWOB' 'smoothRPM' 'smoothTime' 'Pad8UCS' 'UCSmapped' 'dUCSmapped' 'B'
'WOBestslope' 'ROPestslope' 'Testslope' 'NormalBehavior' 'UCSindicator'
'BinnedSmoothBehavior' 'BinnedSmoothUCSIndicator' 'WOBavgslope' 'ROPavgslope'
'Tavgslope' 'SmoothUCSindicator' 'MSEUCSdif' 'UCSvsMSE' 'MSEvsUCS' 'dMSE'
'dsmoothMSE' 'smoothUCSmapped' 'HandClassification' 'UCSMappedSlopeClass'
'LikelyUCS' 'Layer1' 'Layer2' 'Layer1Groups' 'Layer2Groups' 'WOBsloperaw'
```

```
'ROPsloperaw' 'Tsloperaw' 'WOBslopeclassraw' 'ROPslopeclassraw'
'Tslopeclassraw' 'WOBslopeSM' 'ROPslopeSM' 'TslopeSM' 'WOBslopeclassSM'
'ROPslopeclassSM' 'TslopeclassSM' 'PlottedPredictedUCS' 'TimeSeconds'
'timeindicator' 'instantWOBslope' 'instantROPslope' 'instantTslope'
'instWOBclass' 'instROPclass' 'instTclass' 'UCSslopeclassification' 'WOBchar'
'ROPchar' 'Tchar' 'PredictedMSECurve'};
writetable(Pad8Curve1sthalfTable,'C:\Users\Carolyn\Documents\Masters
Research\Spring 2018\Apache\Processed Single
Files\Pad8Curve1sthalf_dysfunction.csv')

Pad8Curve2ndhalfTable=array2table(Pad8Curve2ndhalf);
Pad8Curve2ndhalfTable.Properties.VariableNames={'HoleDepth' 'WeightonBit'
'BitRPM' 'DifferentialPressure' 'RateofPenetration' 'MotorRPM' 'BlockHeight'
'RotaryRPM' 'TotalPumpOutput' 'BitDepth' 'RotaryTorque' 'Inclination' 'Azimuth'
'DysfunctionDepth' 'BitBallingBelief' 'BitBounceBelief' 'StickSlipBelief'
'WhirlBelief' 'OtherDysfunctionBelief' 'NoDysfunctionBelief' 'UCSanalog' 'BHA'
'MSE' 'MSEwithRotaryTorque' 'Time' 'CalcedBitRPM' 'DrillingEfficiency'
'BitAggressiveness' 'HSI' 'Torque' 'TVDSurvey' 'CourseNorthSurvey'
'CourseEastSurvey' 'TVD' 'CourseNorth' 'CourseEast' 'BinnedMSE' 'SmoothMSE'
'dROP' 'dRPM' 'dTorque' 'dWOB' 'dTime' 'dsmoothROP' 'dsmoothRPM'
'dsmoothTorque' 'dsmoothWOB' 'dsmoothTime' 'sWOBandTvsROP' 'ZeroRatio'
'MicroErraticTorque' 'MicroErraticWOB' 'MicroErraticRPM' 'MicroErraticROP'
'TorqueMagnitudeChange' 'WOBMagnitudeChange' 'RPMMagnitudeChange'
'ROPMagnitudeChange' 'sWOBandTandROP' 'sWOBandROPvsT' 'sTandROPvsWOB'
'WOBandTandROP' 'WOBandROPvsT' 'TandROPvsWOB' 'WOBandTvsROP' 'ROPandWOB'
'CoefficientofFriction' 'dROPvdRPM' 'dROPvdT' 'DOC' 'smoothTorque' 'smoothROP'
'smoothWOB' 'smoothRPM' 'smoothTime' 'Pad8UCS' 'UCSmapped' 'dUCSmapped' 'B'
'WOBestslope' 'ROPestslope' 'Testslope' 'NormalBehavior' 'UCSindicator'
'BinnedSmoothBehavior' 'BinnedSmoothUCSIndicator' 'WOBavgslope' 'ROPavgslope'
'Tavgslope' 'SmoothUCSindicator' 'MSEUCSdif' 'UCSvsMSE' 'MSEvsUCS' 'dMSE'
'dsmoothMSE' 'smoothUCSmapped' 'HandClassification' 'UCSMappedSlopeClass'
'LikelyUCS' 'Layer1' 'Layer2' 'Layer1Groups' 'Layer2Groups' 'WOBsloperaw'
'ROPsloperaw' 'Tsloperaw' 'WOBslopeclassraw' 'ROPslopeclassraw'
'Tslopeclassraw' 'WOBslopeSM' 'ROPslopeSM' 'TslopeSM' 'WOBslopeclassSM'
'ROPslopeclassSM' 'TslopeclassSM' 'PlottedPredictedUCS' 'TimeSeconds'
'timeindicator' 'instantWOBslope' 'instantROPslope' 'instantTslope'
'instWOBclass' 'instROPclass' 'instTclass' 'UCSslopeclassification' 'WOBchar'
'ROPchar' 'Tchar' 'PredictedMSECurve'};
writetable(Pad8Curve2ndhalfTable,'C:\Users\Carolyn\Documents\Masters
Research\Spring 2018\Apache\Processed Single
Files\Pad8Curve2ndhalf_dysfunction.csv')
```

X. Create Linear Regression Features for Random Forest: R and Slope

The program presented below generates the R and Slope features over a designated window of each parameter. The program can be run on the full Well 8 curve, or the individual halves. To run on the full curve, “loc2” and “loc1” should be assigned as the file locations of the processed and split Well 8 curve halves. All lines of code within main() should be active and those pertaining to “loc3” should not be active (preceded with a #). The “digits_for_R” should be set to 4. The program then be run again with “loc3” assigned to the file location of the full Well 8 curve data, the “loc3” lines active and the “digits_for_R” set to 5.

```
1.  import pandas as pd
2.  import matplotlib.pyplot as plt
3.  import seaborn as sns
4.  import numpy as np
5.  from sklearn.ensemble import RandomForestClassifier
6.  from scipy import stats
7.  import math as m
8.  from sklearn import tree
9.  from sklearn.tree import export_graphviz
10.
11. def r_and_slope_generator(loc, variable_input, digits_for_r, destination):
12.     """
13.     Partitions columns of the dataframe input through the variable_input into li
    sts of length digits. Computes the r value and slope
14.     (rounded to the nearest tenth) of those lengths (x = 1-
    digits, y = UCS values) and adds them digits_for_r number of times to a new list
    .
15.
16.     If the length of one of the variable_inputs are not evenly divided by digits
    , it will take the remaining part of the list, compute an r value
17.     and slope for them, and return append it the remaining number of times (thus
    , the returned list is the same length as ucs_values).
18.
19.     The r value is multiplied by 10 so it works with SKLearns RandomForestClassi
    fier.
20.
21.     :param loc: input location on device in the form of a string, with backslash
    es replaced with forward slashes.
22.     e.g. C:/Users/name/Desktop/Folder1/something.csv
23.     :param variable_input: what variables to find R values for (list of strings)
    .
```

```
    PRECONDITION: All variables have equal number of data points, i.e. len(variable_input[0]) == len(variable_input[1]) == ...
    == len(variable_input[n])
    PRECONDITION: len(variable_input[0]) > digits
    :param digits_for_r:  number of digits to use for r value (integer)
    :param destination: This is the name of the .csv that the dataframe will be put into. IT OVERWRITES ALL PREVIOUS DATA IN THE CSV. (string)
    :return: Nothing - outputs are put into .csv
    """
    # Importing lists necessary
    output_df = pd.read_csv(loc)
    list_of_vars = []
    counter = 0

    for var in variable_input:
        list_of_vars.append(output_df[var].tolist())
        counter += 1

    # Setting variables required for partitioning the list
    remainder = len(list_of_vars[0]) % digits_for_r
    small_r = m.floor(len(list_of_vars[0]) / digits_for_r) * digits_for_r
    r_values = []
    slopes = []
    x = np.arange(start=1, stop=digits_for_r+1)
    y = []
    subx = np.arange(start=1, stop=remainder+1)
    name_counter = 0
    # Partitioning the list and setting it's values
    for var_list in list_of_vars:
        count = 0
        total_count = 0
        for value in var_list:
            y.append(value)
            count += 1
            total_count += 1
            if count == digits_for_r:
                various_stats = stats.linregress(x, y)
                slope = various_stats[0]
                r_value = various_stats[2]
                rounded_r_value = round(r_value, 1)
                for i in range(digits_for_r):
                    r_values.append(rounded_r_value*10)
                    slopes.append(slope)
                y = []
                count = 0
            if total_count == len(list_of_vars[0]) and small_r != 0:
                various_stats = stats.linregress(subx, y)
                slope = various_stats[0]
                r_value = various_stats[2]
                rounded_r_value = round(r_value, 1)
                for i in range(remainder):
                    r_values.append(rounded_r_value*10)
                    if m.isnan(slope):
                        slopes.append('-999.25')
                    else:
```

```
                        slopes.append(slope)
                y = []
        output_df[f'{variable_input[name_counter]}_R_{digits_for_r}'] = r_values

        output_df[f'{variable_input[name_counter]}_slope_{digits_for_r}'] = slop
es
        slopes = []
        r_values = []
        name_counter += 1
    # Creating a dataframe putting it into a csv
    output_df.to_csv(path_or_buf=destination, index=False)

def main():
    destination1 = 'RandSlope1st.csv'
    loc1 = 'C:/Users/Carolyn/Documents/Masters Research/Python Code/Pad8Curve1st
half_dysfunction.csv'
    destination2 = 'RandSlope2nd.csv'
    loc2 = 'C:/Users/Carolyn/Documents/Masters Research/Python Code/Pad8Curve2nd
half_dysfunction.csv'
    destination3 = 'RandSlopeWholeCurve.csv'
    loc3 = 'C:/Users/Carolyn/Documents/Masters Research/Python Code/Pad8Curve_dy
sfunction.csv'
    digits_for_R = 5
    variable_input = ['smoothWOB', 'smoothROP', 'smoothTorque']
    #variable_input = ['WeightonBit', 'RateofPenetration', 'Torque']

    #r_and_slope_generator(loc1, variable_input, digits_for_R, destination1) #ha
s odd # of values so 4 as window is ok, otherwise x or y size will equal 0 and y
ou cant run regression
    #r_and_slope_generator(loc2, variable_input, digits_for_R, destination2)
          r_and_slope_generator(loc3, variable_input, digits_for_R, destinatio
n3) #have to run this separately because has even # of values, so must use 5 as
window



      if __name__ == "__main__":
          main()
```

Y. Generate Random Forest Models

The program presented below generates the Random Forest models for both the halved and random selection methods for creating the test and train data sets. The code is written to be easily modified to run with smoothed and raw parameter data derivatives. The

Random Forest Classifier function is utilized to generate the model. Additional code has been omitted here that was written to perform a Random Forest Regression model, however a regression was considered not appropriate for the analysis performed.

```
1.  import pandas as pd
2.  import matplotlib.pyplot as plt
3.  import seaborn as sns
4.  import numpy as np
5.  from sklearn.ensemble import RandomForestClassifier
6.  from scipy import stats
7.  import math as m
8.  from sklearn import tree
9.  from sklearn.tree import export_graphviz
10. from sklearn.model_selection import train_test_split
11. from sklearn.metrics import classification_report, confusion_matrix, accuracy_sc
    ore
12. #from sklearn.ensemble import RandomForestRegressor
13. #from sklearn.linear_model import LinearRegression
14. #from sklearn.neural_network import MLPRegressor
15.
16. def import_data(loc, randselect):
17.     """
18.         Gets the relevant data
19.     :param loc: file location on device in the form of a string, with backslashe
    s replaced with forward slashes.
20.     e.g. C:/Users/name/Desktop/Folder1/something.csv
21.     :param regress: True: importing data for a regression algorithm. False: impo
    rting data for a classification algorithm
22.     :return: Pandas dataframe of relevant features
23.     """
24.     df = pd.read_csv(loc)
25.
26.     # Gets relevant features
27.     if randselect is True:
28.         features = df.loc[:, ('smoothWOB_R_5', 'smoothWOB_slope_5', 'smoothROP_R
    _5', 'smoothROP_slope_5', 'smoothTorque_R_5', 'smoothTorque_slope_5', 'instWOBcl
    ass', 'instROPclass', 'instTclass', 'UCSslopeclassification')]
29.         #features = df.loc[:, ('WeightonBit_R_5', 'WeightonBit_slope_5', 'Rateof
    Penetration_R_5', 'RateofPenetration_slope_5', 'Torque_R_5', 'Torque_slope_5', '
    instWOBclass', 'instROPclass', 'instTclass', 'UCSslopeclassification')]
30.
31.     else:
32.         features = df.loc[:, ('smoothWOB_R_4', 'smoothWOB_slope_4', 'smoothROP_R
    _4', 'smoothROP_slope_4', 'smoothTorque_R_4', 'smoothTorque_slope_4', 'instWOBcl
    ass', 'instROPclass', 'instTclass', 'UCSslopeclassification')]
33.         #features = df.loc[:, ('WeightonBit_R_4', 'WeightonBit_slope_4', 'Rateof
    Penetration_R_4', 'RateofPenetration_slope_4', 'Torque_R_4', 'Torque_slope_4', '
    instWOBclass', 'instROPclass', 'instTclass', 'UCSslopeclassification')]
34.
```

```
    return features


def plot(y_var, df, title):
    """
    Very simple plotting function using seaborn.
    :param y_var: The y-variable to plot (string)
    :param df: Dataframe to plot (pd.DataFrame)
    :param title: The title of the plot (string)
    """
    sns.lineplot(x='Number', y=y_var, data=df)
    plt.title(title)
    plt.show()


def variable_importances(rf, feature_list):
    # Get numerical feature importances
    importances = list(rf.feature_importances_)

    # List of tuples with variable and importance
    feature_importances = [(feature, round(importance, 2)) for feature, importan
ce in zip(feature_list, importances)]

    # Sort the feature importances by most important first
    feature_importances = sorted(feature_importances, key=lambda x: x[1], revers
e=True)
    # Print out the feature and importances
    [print('Variable: {:20} Importance: {}'.format(*pair)) for pair in feature_i
mportances]


def visualize_tree(rf, feature_list):
    # Pull out one tree from the forest
    tree = rf.estimators_[5]

    # Export the image to a dot file
    #export_graphviz(tree, out_file='tree.dot', feature_names=feature_list, roun
ded=True, precision=1)
    # i_tree = 0
    # for tree_in_forest in rf.estimators_:
    #     with open('tree_' + str(i_tree) + '.dot', 'w') as my_file:
    #         my_file = tree.export_graphviz(tree_in_forest, out_file=my_file)
    #     i_tree = i_tree + 1

    print(type(tree))
    print(tree.source)
    print(tree)
    # Use dot file to create a graph
    #graph = pydotplus.graph_from_dot_file('tree.dot')

    # Write graph to a png file
    #graph.write_png('tree.png')

 # Split the data (Pad8Curve.csv) into training and testing sets

```

```
def random_forest_classifier_randselect(features, destination):
    """
        Trains random forest classifier using WeightonBit, RateofPenetration, and Torque. Uses randomly selected data as train and test sets.
    :param features: Dataframe of features (pd.DataFrame).
    :param loc: file location on device in the form of a string, with backslashes replaced with forward slashes.
    e.g. C:/Users/name/Desktop/Folder1/something.csv
    :return: Predicted UCS in a list (np.ndarray)
    """

    # Labels are the values we want to predict
    labels = np.array(features['UCSslopeclassification'])

    # Remove the labels from the features axis 1 refers to the columns
    features = features.drop('UCSslopeclassification', axis=1)

            # Saving feature names for later use
            feature_list = list(features.columns)

            # Convert to numpy array
            features = np.array(features)

            indices = np.arange(len(labels))

            train_features, test_features, train_labels, test_labels, indx_train, indx_test = train_test_split(features, labels, indices, test_size = 0.25, random_state = 42)

            clf_rand = RandomForestClassifier(n_estimators=1000, random_state=42)
            clf_rand.fit(train_features, train_labels)


            # Use the forest's predict method on the test data
            predictions_rand = clf_rand.predict(test_features)


            print('RandSelect Confusion Matrix:', confusion_matrix(test_labels,predictions_rand))
            print('RandSelect Classification Report:', classification_report(test_labels,predictions_rand))
            print('RandSelect Accuracy Score:', accuracy_score(test_labels, predictions_rand))

            df = pd.DataFrame(data={'Predictions_RandSelect': predictions_rand, 'Test_Labels': test_labels, 'Test_Set_Index': indx_test})
            df.to_csv(path_or_buf=destination, index=False)

            return predictions_rand, clf_rand, feature_list, test_labels



        def random_forest_classifier_splitinhalf(features, loc, destination):
```

```
            """
                Trains random forest classifier using WeightonBit, RateofPenetration, and Torque. Uses 2nd half of Pad 8 curve as training set, and 1st half as test set.
            :param features: Dataframe of features (pd.DataFrame).
            :param loc: file location on device in the form of a string, with backslashes replaced with forward slashes.
            e.g. C:/Users/name/Desktop/Folder1/something.csv
            :return: Predicted UCS in a list (np.ndarray)
            """
            # Labels are the values we want to predict
            train_labels = np.array(features['UCSslopeclassification'])

            # Remove the labels from the features axis 1 refers to the columns
            features = features.drop('UCSslopeclassification', axis=1)

            # Saving feature names for later use
            feature_list = list(features.columns)

            # Convert to numpy array
            train_features = np.array(features)

            clf = RandomForestClassifier(n_estimators=1000, random_state=42)
            clf.fit(train_features, train_labels)

            ### Grab the test data from Pad8Curve2ndhalf.csv, same way as above

            test_df = import_data(loc, False)
            test_labels = test_df['UCSslopeclassification'].values

            second_half_features = test_df.drop('UCSslopeclassification', axis=1)
            test_features = second_half_features.values

            # Use the forest's predict method on the test data
            predictions = clf.predict(test_features)


            print('SplitinHalf Confusion Matrix:', confusion_matrix(test_labels, predictions))
            print('SplitinHalf Classification Report:', classification_report(test_labels,predictions))
            print('SplitinHalf Accuracy Score:', accuracy_score(test_labels, predictions))

            output_df = pd.read_csv(loc)
            output_df[f'Predictions_SplitinHalf'] = predictions
            output_df.to_csv(path_or_buf=destination, index=False)

            return predictions, clf, feature_list


        def compare_plot(predict, df_actual, randselect):
            """

```

```
179.            :param predict: List of predictions (np.ndarray)
180.            :param df_actual: A dataframe with the values to test against (pd.Da
    taFrame)
181.            :return: Doesn't return anything
182.            """
183.            if randselect is True:
184.                actual_values = df_actual
185.            else:
186.                actual_values = df_actual['UCSslopeclassification']
187.
188.            df = pd.DataFrame(data={'prediction': predict, 'actual': actual_valu
    es, 'Number': np.arange(len(actual_values))})
189.
190.            # Plot the actual values
191.            plt.plot(df['Number'], df['actual'], 'b-', label='actual')
192.
193.            # Plot the predicted values
194.            plt.plot(df['Number'], df['prediction'], 'r--
    ', label='prediction')
195.
196.            plt.xticks(rotation='60')
197.            plt.legend()
198.
199.            # Graph labels
200.            plt.xlabel('Relative Time')
201.            plt.ylabel('UCS Slope Classification')
202.            plt.title('Actual and Predicted Values')
203.
204.            plt.show()
205.
206.
207.        def main():
208.            ############ Runs UCS random forest classification algorithm
209.            #First run the classification using the split in half data
210.            location_of_training_data = 'C:/Users/Carolyn/Documents/Masters Rese
    arch/Python Code/RandSlope2nd.csv'
211.            location_of_testing_data = 'C:/Users/Carolyn/Documents/Masters Resea
    rch/Python Code/RandSlope1st.csv'
212.
213.            features_classification = import_data(location_of_training_data, Fal
    se)
214.
215.            # Creates a new dataframe using the TESTING data and selects out fea
    tures of importance.
216.            features_actual = pd.read_csv(location_of_testing_data)
217.            testing_classification_df = features_actual.loc[:, ('smoothWOB_R_4',
     'smoothWOB_slope_4', 'smoothROP_R_4', 'smoothROP_slope_4', 'smoothTorque_R_4',
    'smoothTorque_slope_4', 'instWOBclass', 'instROPclass', 'instTclass', 'UCSslopec
    lassification')]
218.            #testing_classification_df = features_actual.loc[:, ('WeightonBit_R_
    4', 'WeightonBit_slope_4', 'RateofPenetration_R_4', 'RateofPenetration_slope_4',
     'Torque_R_4', 'Torque_slope_4', 'instWOBclass', 'instROPclass', 'instTclass', '
    UCSslopeclassification')]
219.
220.            # Trains our model
```

```
            Destination_for_Prediction_data1 = '1stHalf_with_Predictions.csv'
            random_forest_clf_model_splitinhalf = random_forest_classifier_split
    inhalf(features_classification, location_of_testing_data, Destination_for_Predic
    tion_data1)

            #print(random_forest_clf_model[0])
            ### Code to plot. Uncomment (remove #) to run. The plot which shows
    will be the last uncommented plotting function.

            #plot('UCSmapped', new_df, "UCS of Pad8Curve2ndhalf")
            #plot('UCSmapped', features, "UCS of Pad8Curve1sthalf")
            compare_plot(random_forest_clf_model_splitinhalf[0], testing_classif
    ication_df, False)
            #visualize_tree(random_forest_clf_model[1], random_forest_clf_model[
    2])
            variable_importances(random_forest_clf_model_splitinhalf[1], random_
    forest_clf_model_splitinhalf[2])

            #Next run the classification using a randomized selection for testin
    g and training set
            location_of_data = 'C:/Users/Carolyn/Documents/Masters Research/Pyth
    on Code/RandSlopeWholeCurve.csv'
            features_classification_rand = import_data(location_of_data, True)
            Destination_for_Prediction_data2 = 'RandSelect_Predictions_with_Test
    Labels.csv'
            random_forest_clf_model_randselect = random_forest_classifier_randse
    lect(features_classification_rand, Destination_for_Prediction_data2)

            compare_plot(random_forest_clf_model_randselect[0], random_forest_cl
    f_model_randselect[3], True)
            variable_importances(random_forest_clf_model_randselect[1], random_f
    orest_clf_model_randselect[2])


        #For reference: 'smoothWOB_R_4', 'smoothWOB_slope_4', 'smoothROP_R_4', '
    smoothROP_slope_4', 'smoothTorque_R_4', 'smoothTorque_slope_4', 'instWOBclass',
    'instROPclass', 'instTclass',


        if __name__ == "__main__":
            main()

```

Z. Compile Random Forest Results with Master Data Set for Analysis

The program presented below combines the test predictions from the Random Forest models with the master Well 8 data set so that further analysis can be performed.

The first program is written to combine the randomly selected test data set with the full Well 8 curve data. The second program is written to combine the 1st half test set with the data from the 1st half of Well 8.

```
%Import Randomly Selected Test set with Predictions, manipulate and add to
%the full curve data set

A=readtable('RandSelect_Predictions_with_TestLabels.csv');
B=readtable('RandSlopeWholeCurve.csv');
Predict=table2array(A);
Data=table2array(B);

n=numel(Data(:,1));
m=numel(Predict(:,1));

Indices=Predict(:,3)+1;

Test_Labels=zeros(n,1);
Predictions=zeros(n,1);

for i=1:m
    index=Indices(i,1);
    Test_Labels(index)=Predict(i,2);
    Predictions(index)=Predict(i,1);
end

Data=[Data Test_Labels Predictions];
Data=array2table(Data);
Data.Properties.VariableNames={'HoleDepth' 'WeightonBit' 'BitRPM'
'DifferentialPressure' 'RateofPenetration' 'MotorRPM' 'BlockHeight' 'RotaryRPM'
'TotalPumpOutput' 'BitDepth' 'RotaryTorque' 'Inclination' 'Azimuth'
'DysfunctionDepth' 'BitBallingBelief' 'BitBounceBelief' 'StickSlipBelief'
'WhirlBelief' 'OtherDysfunctionBelief' 'NoDysfunctionBelief' 'UCSanalog' 'BHA'
'MSE' 'MSEwithRotaryTorque' 'Time' 'CalcedBitRPM' 'DrillingEfficiency'
'BitAggressiveness' 'StickSlipAlarm' 'HSI' 'Torque' 'TVDSurvey'
'CourseNorthSurvey' 'CourseEastSurvey' 'TVD' 'CourseNorth' 'CourseEast'
'BinnedMSE' 'SmoothMSE' 'dROP' 'dRPM' 'dTorque' 'dWOB' 'dTime' 'dsmoothROP'
'dsmoothRPM' 'dsmoothTorque' 'dsmoothWOB' 'dsmoothTime' 'sWOBandTvsROP'
'ZeroRatio' 'MicroErraticTorque' 'MicroErraticWOB' 'MicroErraticRPM'
'MicroErraticROP' 'TorqueMagnitudeChange' 'WOBMagnitudeChange'
'RPMMagnitudeChange' 'ROPMagnitudeChange' 'sWOBandTandROP' 'sWOBandROPvsT'
'sTandROPvsWOB' 'WOBandTandROP' 'WOBandROPvsT' 'TandROPvsWOB' 'WOBandTvsROP'
'ROPandWOB' 'CoefficientofFriction' 'dROPvdRPM' 'dROPvdT' 'DOC' 'smoothTorque'
'smoothROP' 'smoothWOB' 'smoothRPM' 'smoothTime' 'Pad8UCS' 'UCSmapped'
```

```
'dUCSmapped' 'B' 'WOBestslope' 'ROPestslope' 'Testslope' 'NormalBehavior'
'UCSindicator' 'BinnedSmoothBehavior' 'BinnedSmoothUCSIndicator' 'WOBavgslope'
'ROPavgslope' 'Tavgslope' 'SmoothUCSindicator' 'MSEUCSdif' 'UCSvsMSE'
'MSEvsUCS' 'dMSE' 'dsmoothMSE' 'smoothUCSmapped' 'HandClassification'
'UCSMappedSlopeClass' 'LikelyUCS' 'Layer1' 'Layer2' 'Layer1Groups'
'Layer2Groups' 'WOBsloperaw' 'ROPsloperaw' 'Tsloperaw' 'WOBslopeclassraw'
'ROPslopeclassraw' 'Tslopeclassraw' 'WOBslopeSM' 'ROPslopeSM' 'TslopeSM'
'WOBslopeclassSM' 'ROPslopeclassSM' 'TslopeclassSM' 'PlottedPredictedUCS'
'ROPmpers' 'ROPftperhr' 'd' 'TorqueNm' 'Torqueftlbs' 'TimeSeconds'
'timeindicator' 'instantWOBslope' 'instantROPslope' 'instantTslope'
'instWOBclass' 'instROPclass' 'instTclass' 'UCSslopeclassification' 'WOBchar'
'ROPchar' 'Tchar' 'PredictedMSECurve' 'SmoothWOB_R_5' 'SmoothWOB_slope_5'
'SmoothROP_R_5' 'SmoothROP_slope_5' 'SmoothTorque_R_5' 'SmoothTorque_slope_5'
'Test_Labels' 'Predictions'};
writetable(Data,'C:\Users\Carolyn\Documents\Masters Research\Spring
2018\Apache\Processed Single Files\Pad8CurvewithPredictions_RandSelect.csv')

A=readtable('1stHalf_with_Predictions.csv');
B=readtable('RandSlope1st.csv');
Predict=table2array(A);
Data=table2array(B);

n=numel(Data(:,1));
m=numel(Predict(:,1));

Indices=Predict(:,3)+1;

Test_Labels=zeros(n,1);
Predictions=zeros(n,1);

for i=1:m
    index=Indices(i,1);
    Test_Labels(index)=Predict(i,2);
    Predictions(index)=Predict(i,1);
end

Data=[Data Test_Labels Predictions];
Data=array2table(Data);
Data.Properties.VariableNames={'HoleDepth' 'WeightonBit' 'BitRPM'
'DifferentialPressure' 'RateofPenetration' 'MotorRPM' 'BlockHeight' 'RotaryRPM'
'TotalPumpOutput' 'BitDepth' 'RotaryTorque' 'Inclination' 'Azimuth'
'DysfunctionDepth' 'BitBallingBelief' 'BitBounceBelief' 'StickSlipBelief'
'WhirlBelief' 'OtherDysfunctionBelief' 'NoDysfunctionBelief' 'UCSanalog' 'BHA'
'MSE' 'MSEwithRotaryTorque' 'Time' 'CalcedBitRPM' 'DrillingEfficiency'
'BitAggressiveness' 'StickSlipAlarm' 'HSI' 'Torque' 'TVDSurvey'
'CourseNorthSurvey' 'CourseEastSurvey' 'TVD' 'CourseNorth' 'CourseEast'
```

```
'BinnedMSE' 'SmoothMSE' 'dROP' 'dRPM' 'dTorque' 'dWOB' 'dTime' 'dsmoothROP'
'dsmoothRPM' 'dsmoothTorque' 'dsmoothWOB' 'dsmoothTime' 'sWOBandTvsROP'
'ZeroRatio' 'MicroErraticTorque' 'MicroErraticWOB' 'MicroErraticRPM'
'MicroErraticROP' 'TorqueMagnitudeChange' 'WOBMagnitudeChange'
'RPMMagnitudeChange' 'ROPMagnitudeChange' 'sWOBandTandROP' 'sWOBandROPvsT'
'sTandROPvsWOB' 'WOBandTandROP' 'WOBandROPvsT' 'TandROPvsWOB' 'WOBandTvsROP'
'ROPandWOB' 'CoefficientofFriction' 'dROPvdRPM' 'dROPvdT' 'DOC' 'smoothTorque'
'smoothROP' 'smoothWOB' 'smoothRPM' 'smoothTime' 'Pad8UCS' 'UCSmapped'
'dUCSmapped' 'B' 'WOBestslope' 'ROPestslope' 'Testslope' 'NormalBehavior'
'UCSindicator' 'BinnedSmoothBehavior' 'BinnedSmoothUCSIndicator' 'WOBavgslope'
'ROPavgslope' 'Tavgslope' 'SmoothUCSindicator' 'MSEUCSdif' 'UCSvsMSE'
'MSEvsUCS' 'dMSE' 'dsmoothMSE' 'smoothUCSmapped' 'HandClassification'
'UCSMappedSlopeClass' 'LikelyUCS' 'Layer1' 'Layer2' 'Layer1Groups'
'Layer2Groups' 'WOBsloperaw' 'ROPsloperaw' 'Tsloperaw' 'WOBslopeclassraw'
'ROPslopeclassraw' 'Tslopeclassraw' 'WOBslopeSM' 'ROPslopeSM' 'TslopeSM'
'WOBslopeclassSM' 'ROPslopeclassSM' 'TslopeclassSM' 'PlottedPredictedUCS'
'ROPmpers' 'ROPftperhr' 'd' 'TorqueNm' 'Torqueftlbs' 'TimeSeconds'
'timeindicator' 'instantWOBslope' 'instantROPslope' 'instantTslope'
'instWOBclass' 'instROPclass' 'instTclass' 'UCSslopeclassification' 'WOBchar'
'ROPchar' 'Tchar' 'PredictedMSECurve' 'SmoothWOB_R_5' 'SmoothWOB_slope_5'
'SmoothROP_R_5' 'SmoothROP_slope_5' 'SmoothTorque_R_5' 'SmoothTorque_slope_5'
'Test_Labels' 'Predictions'};
writetable(Data,'C:\Users\Carolyn\Documents\Masters Research\Spring
2018\Apache\Processed Single Files\Pad8CurvewithPredictions_1sthalf.csv')
%Combine the test predictions from the 1st half of the curve with the
%master data set for the 1st half of the curve
A=readtable('1stHalf_with_Predictions.csv');
B=readtable('RandSlope1st.csv');
Predict=table2array(A);
Data=table2array(B);

n=numel(Data(:,1));
m=numel(Predict(:,1));

Indices=Predict(:,3)+1;

Test_Labels=zeros(n,1);
Predictions=zeros(n,1);

for i=1:m
    index=Indices(i,1);
    Test_Labels(index)=Predict(i,2);
    Predictions(index)=Predict(i,1);
end

Data=[Data Test_Labels Predictions];
```

```
Data=array2table(Data);
Data.Properties.VariableNames={'HoleDepth' 'WeightonBit' 'BitRPM'
'DifferentialPressure' 'RateofPenetration' 'MotorRPM' 'BlockHeight' 'RotaryRPM'
'TotalPumpOutput' 'BitDepth' 'RotaryTorque' 'Inclination' 'Azimuth'
'DysfunctionDepth' 'BitBallingBelief' 'BitBounceBelief' 'StickSlipBelief'
'WhirlBelief' 'OtherDysfunctionBelief' 'NoDysfunctionBelief' 'UCSanalog' 'BHA'
'MSE' 'MSEwithRotaryTorque' 'Time' 'CalcedBitRPM' 'DrillingEfficiency'
'BitAggressiveness' 'StickSlipAlarm' 'HSI' 'Torque' 'TVDSurvey'
'CourseNorthSurvey' 'CourseEastSurvey' 'TVD' 'CourseNorth' 'CourseEast'
'BinnedMSE' 'SmoothMSE' 'dROP' 'dRPM' 'dTorque' 'dWOB' 'dTime' 'dsmoothROP'
'dsmoothRPM' 'dsmoothTorque' 'dsmoothWOB' 'dsmoothTime' 'sWOBandTvsROP'
'ZeroRatio' 'MicroErraticTorque' 'MicroErraticWOB' 'MicroErraticRPM'
'MicroErraticROP' 'TorqueMagnitudeChange' 'WOBMagnitudeChange'
'RPMMagnitudeChange' 'ROPMagnitudeChange' 'sWOBandTandROP' 'sWOBandROPvsT'
'sTandROPvsWOB' 'WOBandTandROP' 'WOBandROPvsT' 'TandROPvsWOB' 'WOBandTvsROP'
'ROPandWOB' 'CoefficientofFriction' 'dROPvdRPM' 'dROPvdT' 'DOC' 'smoothTorque'
'smoothROP' 'smoothWOB' 'smoothRPM' 'smoothTime' 'Pad8UCS' 'UCSmapped'
'dUCSmapped' 'B' 'WOBestslope' 'ROPestslope' 'Testslope' 'NormalBehavior'
'UCSindicator' 'BinnedSmoothBehavior' 'BinnedSmoothUCSIndicator' 'WOBavgslope'
'ROPavgslope' 'Tavgslope' 'SmoothUCSindicator' 'MSEUCSdif' 'UCSvsMSE'
'MSEvsUCS' 'dMSE' 'dsmoothMSE' 'smoothUCSmapped' 'HandClassification'
'UCSMappedSlopeClass' 'LikelyUCS' 'Layer1' 'Layer2' 'Layer1Groups'
'Layer2Groups' 'WOBsloperaw' 'ROPsloperaw' 'Tsloperaw' 'WOBslopeclassraw'
'ROPslopeclassraw' 'Tslopeclassraw' 'WOBslopeSM' 'ROPslopeSM' 'TslopeSM'
'WOBslopeclassSM' 'ROPslopeclassSM' 'TslopeclassSM' 'PlottedPredictedUCS'
'ROPmpers' 'ROPftperhr' 'd' 'TorqueNm' 'Torqueftlbs' 'TimeSeconds'
'timeindicator' 'instantWOBslope' 'instantROPslope' 'instantTslope'
'instWOBclass' 'instROPclass' 'instTclass' 'UCSslopeclassification' 'WOBchar'
'ROPchar' 'Tchar' 'PredictedMSECurve' 'SmoothWOB_R_5' 'SmoothWOB_slope_5'
'SmoothROP_R_5' 'SmoothROP_slope_5' 'SmoothTorque_R_5' 'SmoothTorque_slope_5'
'Test_Labels' 'Predictions'};
writetable(Data,'C:\Users\Carolyn\Documents\Masters Research\Spring
2018\Apache\Processed Single Files\Pad8CurvewithPredictions_1sthalf.csv”
```

Glossary

A_{bit} = Bit Area

CCS = Confined Compressive Strength

CFM = Continuous Fracture Model

DE = Drilling Efficiency

DOC = Depth of Cut

DSE = Drilling Specific Energy

FDP = Fast Drill Process (ExxonMobil)

ft = Feet

gal = Gallons

GR = Gamma Ray

HHP_{bit} = Hydraulic Horsepower at the Bit

HSI = Hydraulic Horsepower per Square Inch

ISIP = Instantaneous Shut-in Pressure

K_n = Revolutions per Gallon

LWD = Logging While Drilling

MD = Measured Depth

MSE = Mechanical Specific Energy

N = Rotary RPM

NFI = Natural Fracture Index

psi = Pounds per Square Inch

Q = Flow Rate

ROP = Rate of Penetration

RPM = Revolutions per Minute

TD = Total Depth

T_{max} = Maximum Torque

UCS = Unconfined Compressive Strength

ΔP = Differential Pressure

ΔP_{max} = Maximum Differential Pressure

λ = Bit Design Constant

References

Alsubaih, A., Albadran, F., & Alkanaani, N. (2018). Mechanical Specific Energy and Statistical Techniques to Maximizing the Drilling Rates for Production Section of Mishrif Wells in Southern Iraq Fields. *International Association of Drilling Contractors and Society of Petroleum Engineers Middle East Drilling Technology Conference and Exhibition.* Abu Dhabi.

Armenta, M. (2008). Identifying Inefficient Drilling Conditions Using Drilling-Specific Energy. *Annual Technical Conference and Exhibition of the Society of Petroleum Engineers.* Denver.

Breiman, L. (1994). *Bagging Predictors.* Berkeley: Department of Statistics University of California.

Breiman, L. (2001). *Random Forests.* Berkeley: Statistics Department University of California.

Donges, N. (2018, February 22). *The Random Forest Algorithm*. Retrieved from Towards Data Science: https://towardsdatascience.com/the-random-forest-algorithm-d457d499ffcd

Dupriest, F. E., & Koederitz, W. L. (2005). Maximizing Drill Rates with Real-Time Surveillance of Mechanical Specific Energy. *International Association of Drilling Contractors and Society of Petroleum Engineers Drilling Conference.* Amsterdam.

Dupriest, F., Witt, J., & Remmert, S. (2005). Maximizing ROP With Real-Time Analysis of Digital Data and MSE. *International Petroleum Technology Conference.* Doha.

Hareland, G., & Hoberock, L. (1993). Use of Drilling Parameters to Predict In-Situ Stress Bounds. *International Association of Drilling Contractors and Society of Petroleum Engineers Drilling Conference.* Amsterdam.

Hareland, G., & Nygaard, R. (2007). Calculating Unconfined Rock Strength from Drilling Data. *American Rock Mechanics Association US Rock Mechanics Symposium*, (pp. 27-31). Vancouver.

Hareland, G., & Rampersad, P. (1994). Drag-Bit Model Including Wear. *Society of Petroleum Engineers Latin America/Caribbean Petroleum Engineering Conference.* Buenos Aires.

Islam, N., Vijapurapu, R., Jones, M., McLennan, J., Moore, J., Rickard, W., . . . Vagnetti, R. (2018). Application of Mechanical Specific Energy and At-the-Bit Measurements for Geothermal Drilling Applications in Hot, High Strength, High Modulus Reservoirs. *American Rock Mechanics Association US Rock Mechanics and Geomechanics Symposium.* Seattle.

Jacques, A., Ouenes, A., Dirksen, R., Paryani, M., Rehman, S., & Bari, M. (2017). Completion Optimization While Drilling - Geomechanical Steering Towards Fracable Rock Using Corrected Mechanical Specific Energy. *Unconventional Resources Technology Conference.* Austin.

Kerkar, P. B., Hareland, G., Fonseca, E. R., & Hackbarth, C. J. (2014). Estimation of Rock Compressive Strength Using Downhole Weight-on-Bit and Drilling Models. *International Petroleum Technology Conference.* Doha.

Kuru, E., & Wojtanowsicz, A. (1988). A Method for Detecting In-Situ PDC Bit Dull and Lithology Change. *International Association of Drilling Contractors and Society of Petroleum Engineers Drilling Conference.* Dallas.

Logan, W. (2015). Engineered Shale Completions Based on Common Drilling Data. *Society of Petroleum Engineers Annual Technical Conference and Exhibition.* Houston.

Majidi, R., Albertin, M., & Last, N. (2016). Method for Pore Pressure Estimation Using Mechanical Specific Energy and Drilling Efficiency. *International Association of Drilling Contractors and Society of Petroleum Engineers Drilling Conference and Exhibition.* Fort Worth.

Ouenes, A., Dirksen, R., Paryani, M., Rehman, S., & Bari, M. (2017). Completion Optimization While Drilling - Geomechanical Steering Towards Fracable Rock for Optimal Selection of Stage Spacing and Cluster Density in Unconventional Wells. *Society of Petroleum Engineers Kingdom of Saudi Arabia Annual Technical Symposium and Exhibition.* Dammam.

Pessier, R., & Fear, M. (1992). Quantifying Common Drilling Problems with Mehcanical Specific Energy and a Bit-Specific Coefficient of Sliding Friction. *Annual Technical Conference and Exhibition of the Society of Petroleum Engineers.* Washington, DC.

Pinto, C., & Lima, A. (2016). Mechanical Specific Energy for Drilling Optimization in Deepwater Brazilian Salt Environments. *International Association of Drilling Contractors and Society of Petroleum Engineers Asia Pacific Drilling Technology Conference.* Singapore.

Rabia, H. (1985). Specific Energy as a Criterion for Bit Selection. *Journal of Petroleum Technology*, 1225-1229.

Richard, T., Detournay, E., Fear, M., Miller, B., Clayton, R., & Matthews, O. (2002). Influence of Bit-Rock Interaction on Stick-Slip Vibrations of PDC Bits. *Society of Petroleum Engineers Annual Technical Conference and Exhibition.* San Antonio.

Ronaghan, S. (2018, May 11). *The Mathematics of Decision Trees, Random Forest and Feature Importance in Scikit-learn and Spark.* Retrieved from Medium: https://medium.com/@srnghn/the-mathematics-of-decision-trees-random-forest-and-feature-importance-in-scikit-learn-and-spark-f2861df67e3

Skinner, M. D., Van Domelen, M., & Grieser, B. (2016). Utilizing Measured Drilling Parameters to Optimize Completions Design. *Unconventional Resources Technology Conference.* San Antonio.

Teale, R. (1965). The Concept of Specific Energy in Rock Drilling. *International Journal of Rock Mechanics and Mining Sciences*, 57-73.

Walstrom, J. E., Harvey, R. P., & Eddy, H. D. (1972). A Comparison of Various Directional Survey Models and an Approach to Model Error Analysis. *Journal of Petroleum Technology*, 935-943.

Warren, T. (1984). Factors Affecting Torque for a Roller Cone Bit. *Journal of Petroleum Technology*, 1500-1508.

Waughman, R. J., Kenner, J. V., & Moore, R. A. (2002). Real-Time Specific Energy Monitoring Reveals Drilling Inefficiency and Enhances the Understanding of When to Pull Worn PDC Bits. *International Association of Drilling Contractors and Society of Petroleum Engineers Drilling Conference.* Dallas.

Zhao Fei, Wang Haige, Cui Meng, Zhang Huabei, & Pang Huiwen. (2017). Monitoring and Mitigating Downhole Vibration with MSE in XinJiang Oil Field of China. *American Rock Mechanics Association US Rock Mechanics and Geomechanics Symposium.* San Francisco.

Zhou, Y., Zhang, W., Gamwo, I., Lin, J., Eastman, H., Gill, M., & Whipple, G. (2012). Mechanical Specific Energy Versus Depth of Cut. *American Rock Mechanics Association US Rock Mechanics and Geomechanics Symposium.* Chicago.

Vita

Carolyn Powell graduated from the University of Texas at Austin in May 2019 with a Master of Science in Petroleum Engineering. She previously graduated from the University of Texas at Austin in May 2013 with a Bachelor of Science in Mechanical Engineering. Prior to completing her graduate studies, she worked for BP as a Subsea Engineer in the Gulf of Mexico. Upon graduating with her masters, she joins Equinor as a Drilling Engineer, working on their North American onshore assets.

Email: clpowell129@gmail.com

This dissertation was typed by Carolyn Powell.